Don't Blame Developers! Examining a Password-Storage Study Conducted with Students, Freelancers, and Company Developers

Dissertation
zur
Erlangung des Doktorgrades (Dr. rer. nat.)
der
Mathematisch-Naturwissenschaftlichen Fakultät
der
Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von
Alena Naiakshina
aus
Frunse, Kirgisistan

Bonn, Juli 2020

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Rheinischen Friedrich-Wilhelms-Universität Bonn

1. Gutachter:	Prof. Dr. Matthew Smith
2. Gutachterin:	Dr. Katharina Krombholz
Tag der Promotion:	19.11.2020
Erscheinungsjahr:	2020

Abstract

Authentication systems are a major concern for the usable security and privacy community. Twenty years ago, the seminal work “Users Are Not The Enemy” [1] by Adams and Sasse initiated a user-centred approach to research. This work was followed by extensive research on end users’ security behavior around authentication systems, which resulted in many suggestions for improvement. Many of these proposals were passed on to software developers, who were considered experts who should know better and were expected to address the issues. However, the related work “Developers Are Not The Enemy” [2] by Green and Smith, citing Adams and Sasse [1], points out that, similar to end users, developers are usually not security experts. Thus, they also struggle with usability and security issues. This is highlighted by the high number of reported security breaches that have compromised millions of end-user passwords. In fact, much of the work invested in usable authentication systems might be in vain if software developers fail to securely store user passwords in databases.

Motivated by the recent security breaches, this thesis aims to provide deeper insights into developers’ security behavior and to clarify why software developers so often fail to store user passwords securely. Therefore, this thesis describes a Java-based password-storage study that was conducted with different samples of developers: computer science (CS) students, freelancers, and professional developers employed by different companies. Participants were instructed to complete the registration functionalities for a social network platform. In order to investigate whether software developers think about security without prompting, half the participants were told the study was about application programming interface (API) usability (non-prompted for security), while the other half were specifically instructed to securely store user passwords (prompted for security). The study also investigated whether an API’s level of password storage security support affects developers’ security behavior. Thus, half the participants used a framework offering opt-in support for secure password storage (Spring), while the other half were provided a programming frame with JavaServer Faces (JSF), which required them to implement password storage security without support.

Initially, a qualitative and a quantitative study were conducted with 20/40 CS students from the University of Bonn in a laboratory setting. The most important finding of this study was that all the students who were not prompted for security submitted solutions in which user passwords were stored in plaintext in the database. Furthermore, a number of the prompted participants who considered password storage security still chose weak security practices. However, some participants claimed that they would have stored user passwords securely in the database if they were solving the task for a real company. In order to test whether these findings were a study artifact, a follow-up study was conducted with 43 freelancers recruited via Freelancer.com. A pilot study with the freelancers suggested that a university context might lead them to believe that university students were hiring them to do their homework. Therefore, this time, the freelancers were not informed that the study was conducted by a research team. Instead, they were told that they were working for a start-up that had lost recently a developer from their team. In this study, freelancers behaved similarly to students with regard to user password storage. Freelancers also had often misconceptions about secure password storage and chose weak practices. Finally, 36 company developers were invited to take part in a password-storage study. They were recruited through their companies and also via the German business social platform Xing. Company developers submitted significantly more secure solutions than students, and they also chose significantly better password storage parameters than students or freelancers. Thus, in absolute terms, they performed better than students and freelancers. However, in relative terms, the results were similar: Security prompting and framework had a significant effect on password storage security for all samples. When prompted, more participants submitted secure solutions, and participants made better parameter choices for password storage security when they used Spring instead of JSF (freelancers used only JSF and thus were not tested for the variable framework).

Additionally, the four studies offered insights for the ecological validity of security studies with developers. The student studies provided an early indication that qualitative research might reveal essential insights without the need to conduct quantitative studies for specific use cases. Furthermore, if the usable security and privacy community is more interested in how security systems can be improved than in which developer group performs best, it might be valid to conduct studies with students rather than professionals. What is more, freelancers tended to behave similarly to students with regard to secure password storage, although they were not aware of the purpose of the study. This suggests that participants tended to ignore the security aspects of software even when working on a web application intended for the use in the real world.

Based on these findings, this thesis has made some recommendations for improving password storage security and regarding the methodological implications for security studies with developers.

Zusammenfassung

Authentifizierungssysteme sind ein wichtiges Anliegen der “Usable Security and Privay” Gemeinschaft. Vor 20 Jahren stießen Adams und Sasse [1] mit ihrer bahnbrechenden Arbeit “Users Are Not The Enemy” einen nutzerzentrierten Forschungsansatz an. Es folgte eine umfangreiche Erforschung von sicherheitsrelevantem Verhalten von Endnutzern mit Authentifizierungssystemen, die viele Verbesserungsansätze mich sich brachte. Viele dieser Ansätze wurden an Softwareentwickler weitergereicht, die “es besser wissen” und diese Ansätze umzusetzen sollten. Mit ihrer Arbeit “Developers Are Not The Enemy” [2] verweisen Green und Smith auf Adams und Sasse [1] und stellen heraus, dass Entwickler, genau wie Endnutzer, üblicherweise keine Sicherheitsexperten sind. Sie haben ebenfalls Schwierigkeiten diverse sicherheitsrelevante Aufgaben umzusetzen. Diese Tatsache spiegelt sich auch in den vielen Sicherheitslücken wider, aufgrund derer Millionen von Endnutzerpasswörtern offen gelegt wurden. Tatsächlich ist die viele Arbeit, die in die Erforschung von Authentifizierungssystemen investiert wurde vergeblich, wenn Softwareentwickler daran scheitern Nutzerpasswörter sicher in Datenbanken zu speichern.

Motiviert durch die kürzlich bekannt gewordenen Sicherheitslücken, zielt diese Arbeit darauf ab tiefere Einblicke in das Sicherheitsverhalten von Entwicklern zu gewähren und zu erörtern warum Softwareentwickler Nutzerpasswörter so oft unsicher speichern. Dafür beschreibt diese Arbeit eine Java-basierte Passwortstudie, die mit unterschiedlichen Entwicklern durchgeführt wurde: Informatikstudenten, Freiberuflern und professionellen Entwicklern, die bei unterschiedlichen Unternehmen beschäftigt sind. Die Teilnehmer wurden dazu aufgefordert eine Registrierungsfunktion für ein soziales Netzwerk zu vervollständigen. Um zu untersuchen, ob Softwareentwickler ohne Aufforderung an Sicherheit denken, wurde der Hälfte der Teilnehmer mitgeteilt, dass die Studie lediglich die Benutzbarkeit einer Anwendungsschnittstelle (engl. application programming interface (API)) erforscht, während die andere Hälfte explizit dazu aufgefordert wurde Nutzerpasswörter sicher zu speichern. Zusätzlich dazu wurde ebenfalls untersucht, ob APIs, die ein unterschiedliches Level an Hilfestellung für das sichere Speichern von Passwörtern anbieten, das sicherheitsrelevante Verhalten von Entwicklern beeinflussen. Die eine Hälfte der Teilnehmer hat daher ein Framework genutzt, das Hilfestellung zum sicheren Passwortspeichern anbietet (Spring), während die andere Hälfte einen Programmierrahmen bekamen, der auf das Framework JavaServer Faces (JSF) basierte und keine Hilfestellung zum sicheren Passwortspeichern bot.

Zunächst wurde eine qualitative und eine quantitative Studie mit 20/40 Informatikstudenten der Universität Bonn in einer Laborumgebung durchgeführt. Die wichtigste Erkenntnis dieser Studie

zeigte auf, dass Teilnehmer, die nicht explizit dazu aufgefordert worden sind Nutzerpasswörter sicher zu speichern, Lösungen einreichten, in denen die Nutzerpasswörter im Klartext in der Datenbank gespeichert wurden. Darüber hinaus haben mehrere Teilnehmer, die zum sicheren Passwortspeichern aufgefordert worden sind, schwache Sicherheitsverfahren implementiert. Einige Teilnehmer behaupteten jedoch, dass sie die Nutzerpasswörter sicher gespeichert hätten, wenn die Aufgabe von einem echten Unternehmen gestellt worden wäre. Um zu erforschen, ob die Ergebnisse ein Artefakt darstellen, wurde eine anschließende Studie mit 43 freiberuflichen Entwicklern durchgeführt, die auf Freelancer.com rekrutiert worden sind. Eine auf dieser Plattform durchgeführte Pilotstudie suggerierte jedoch, dass die Freiberufler glauben könnten, sie würden von Universitätsstudenten engagiert werden, um ihre Hausaufgaben für sie zu erledigen. Daher wurden die Freiberufler für diese Studie nicht darüber in Kenntnis gesetzt, dass die Studie von einer Wissenschaftsgruppe durchgeführt wird. Stattdessen wurde ihnen mitgeteilt, dass sie für ein Start-up beschäftigt werden, das vor kurzem einen Entwickler verloren hat. Im Hinblick auf das sichere Passwortspeichern zeigten die Freiberufler ein ähnliches Verhalten zu dem der Studenten auf. Die Freiberufler zeigten Missverständnisse im Hinblick auf das sichere Passwortspeichern auf und wählten oft schwache Sicherheitsverfahren. Schließlich wurden 36 Entwickler zu einer Passwortstudie eingeladen, die bei unterschiedlichen Unternehmen beschäftigt waren. Diese wurden über ihre Unternehmen sowie das deutsche Geschäftsnetzwerk Xing rekrutiert. Die Entwickler aus den Unternehmen reichten signifikant mehr sicherer Lösungen ein als Studenten. Sie wählten auch signifikant bessere Parameter zum sicheren Passwortspeichern als Informatikstudenten oder freiberufliche Entwickler. In absoluter Hinsicht haben Entwickler aus Unternehmen daher besser abgeschnitten als Studenten oder Freiberufler. In relativer Hinsicht waren die Ergebnisse jedoch vergleichbar: die Aufforderung zum sicheren Passwortspeichern sowie das Framework hatten in allen Gruppen einen signifikanten Effekt. Mit der sicherheitsrelevanten Aufforderung haben mehr Teilnehmer sichere Lösungen eingereicht. Zudem haben Teilnehmer bessere Parameter zum sicheren Passwortspeichern gewählt, wenn sie Spring anstatt JSF nutzten (die Freiberufler haben lediglich JSF genutzt und wurden daher nicht für das Framework getestet).

Ferner gewähren die vier Studien Einblicke in die Ökologische Validität (engl. ecological validity) von sicherheitsrelevanten Studien mit Entwicklern. Die Studien mit den Studenten zeigten erste Anzeichen dafür auf, dass qualitative Studien bereits essentielle Einblicke in Forschungsergebnisse liefern können und somit quantitative Studien in speziellen Fällen möglicherweise nicht vonnöten sind. Zudem, insofern die "Usable Security and Privay" Gemeinschaft daran interessiert ist zu erforschen wie Sicherheitssysteme verbessert werden können anstatt die Leistung von Entwicklergruppen zu vergleichen, könnte es valide sein Studien mit Studenten anstatt mit professionellen Entwicklern durchzuführen. Darüber hinaus zeigten Freiberufler ein ähnliches Verhalten wie die Informatikstudenten im Hinblick auf das sichere Passwortspeichern auf, obwohl ihnen nicht bewusst war, dass sie an einer Studie teilnahmen. Dieses Verhalten deutet daraufhin, dass Teilnehmer dazu neigen Sicherheitsaspekte von Software auch dann zu ignorieren, wenn sie an einer Webanwendung

arbeiten, die für den Gebrauch in der “echten Welt” gedacht ist.

Basierend auf diesen Erkenntnissen macht diese Arbeit Vorschläge dafür wie das sichere Passwortspeichern verbessert werden könnte und liefert eine Diskussionsgrundlage für methodologische Implikationen für sicherheitsrelevante Studien mit Entwicklern.

Acknowledgments

First of all, I would like to thank my supervisor and mentor, Prof. Dr. Matthew Smith, who provided me with the incredible opportunity to work in this very interesting research field and who has always assisted me with his invaluable advice and continuous support. Thanks to him and his trust, I have been able to grow every day and to improve my skills, which has helped me become a better researcher. I would also like to thank my second examiner, Dr. Katharina Krombholz, for contributing the time and effort to provide me feedback on this thesis. I highly appreciate her advice and invaluable encouragement for future research opportunities.

Additionally, I would like to thank all my co-authors, colleagues, and research assistants. Without their help, much of this research would not have been possible. In particular, I would like to thank my long-time friend and teammate Anastasia Danilova, who has always assisted me and invested her valuable time and effort to improve our research. Her invaluable support and encouragement motivated me to conduct all these studies. My special thanks go also to Christian Tiefenau and Eva Gerlitz, who invested time and effort to assist me with different studies.

I would like to thank my wonderful mother, Elena; my remarkable sister, Anastasia; and the rest of my incredible family and friends for their advice and support throughout this process. My special thanks go to my beloved partner, Markus, who has always inspired me, supported me, and encouraged me to become a better version of myself. Finally, I dedicate this thesis to my mother, who made my life and this journey possible.

Contents

1. Introduction

1.1. Motivation

With their pioneering work “Users Are Not The Enemy” [1] from 1999, Adams and Sasse introduced a new way of thinking to the computer science community by calling for the adoption of a user-centered approach to the design of security mechanisms. The concept of “user-centered security” [3] was introduced by Zurko and Simon in 1996. Around the same time, Whitten and Tygar published “Why Johnny Can’t Encrypt” [4], which addresses user interfaces for message encryption, showing that security software that lacks usability can lead to fatal security errors. These fundamental papers created the foundation of the usable security and privacy community [2]. For about 20 years, research has focused on improving the usability of security and privacy preserving systems for end users (e.g., [5–40]). Issues in this area are often passed on to software developers and administrators, who seem to frequently be held responsible for unusable security.

In “Developers Are Not The Enemy” [2], Green and Smith cite Adams and Sasse [1] highlighting that not only end users but also software developers and administrators struggle with unusable security software. These parties’ tasks include the administration, configuration, and development of secure software. Like end users, developers and administrators are usually not security experts and also struggle with security related tasks [2]. While it is important to study end user’s security behavior, it is also essential to understand that administrators and developers are themselves end users of tools, software, libraries, and application programming interfaces (APIs) they use to carry out security-related tasks. Therefore, these parties should also be considered in research on usable security and privacy solutions. However, compared to the amount of research on the needs and perceptions of end users, only a handful of studies on usable security investigate the needs and perceptions of administrators (e.g., [20, 41–47]) or developers (e.g., [48–54]). Therefore, in the following, I will outline the need to extend existing research to include these underrepresented research parties.

In existing end-user research, one important research field is dedicated to the exploration and improvement of authentication systems and end users’ password behavior (e.g., [1, 55–93]). However, while individual users are responsible for creating and storing their own passwords, the developers of applications that request user credentials are responsible for securely storing large amounts of end-user passwords. As several security breaches have demonstrated (e.g., [57, 58, 94, 95]), this is not a trivial task. Green and Smith argue that since software developers often rely on APIs and are rarely security experts, designers and cryptographers should keep usability in mind when developing cryptographic (crypto) APIs and libraries. Green and Smith suggest that research on usable security needs to be extended to support developers who have to interact with complex security APIs when, for instance, ensuring secure password storage.

Developers must consider that adversaries could attack users' passwords on the front end of web service presentation (online attacks) or on the server-side/back end (offline attacks) [95]. While there are ways to limit user login attempts to prevent online attacks, such as CAPTCHA [96] and temporary account lock-outs, Florêncio and Herley [95] demonstrate that offline attacks, which are the justification for the demands placed on users to create strong passwords, are not as common as is generally believed. This follows from the fact that they are only needed if leaked user passwords are securely stored by salting and hashing them [95, 97]. However, recent large-scale breaches have shown that developers struggle with this task [57, 58, 94, 95]. For example, Bonneau and Preibusch [98] analyzed password security strategies of 150 websites and found that 25% of these websites emailed plaintext passwords to users as to restore access in case of a lost or forgotten password. Obviously, they were storing user passwords in plaintext.

In 2007, Florêncio and Herley [99] found that most users have an average of 6.5 passwords but use about 25 applications requiring passwords. In 2016, Wash et al. [89] found that users tend to reuse each password on 1.7 to 3.4 different websites. Nowadays, the number of accounts is increasing at a "14% rate, meaning it doubles every 5 years" [100]. In 2020, the average number of accounts per Internet user is estimated at 207 [100]. Unsurprisingly, users tend to reuse passwords, regardless of the security policies and measures implemented by different websites [89, 99, 101]. This is especially true for those passwords that must be entered frequently [89]. However, an attacker who obtains access to a database of login credentials including plaintext passwords from a web service with no security measures can use those passwords to access sensitive information about users who are using the same passwords on web services with high security standards. Consequently, passwords that are reused on web services with little or no security pose a risk to the sensitive data stored by companies with high security standards.

Since security breaches expose millions of user passwords every year [94, 95, 102, 103] and storing and authenticating user login data is one of the most critical security tasks for software developers [104], this thesis addresses several research questions about password storage security:

- Do developers struggle with user password storage security, and if so, why? The examination of this question will also help to understand why security breaches are observed so often "in the wild."
- How do developers securely store user passwords in databases and what security practices do they follow?
- Do developers think about security on their own, or do they need specific instructions to consider it?

- How do information sources influence developers' security behavior?
- Do APIs that offer security support help developers meet higher security standards?
- Is user password storage security affected by whether developers are students or professionals?

This thesis will investigate developers' security behavior around password-storage on a *primary level*. For this examination, usability studies with developers must be conducted. While there are ample security studies with end users, little is known about appropriately designing and conducting security studies with software developers. For instance, are lab studies required, or are online studies sufficient? Should qualitative or quantitative methods be used? Can the findings of studies conducted with students be generalized, or is it necessary to recruit professionals to obtain generalizable findings? Acar et al. [105] outline these and further issues that need to be addressed to ensure high-quality usability security research with developers. One major concern about user studies in general is their ability to reflect the real world, which is referred to as *ecological validity* [105, 106]. Conducting usability security studies with developers is especially challenging for several reasons. Due to time and financial constraints, it is extremely difficult to recruit enough professionals to obtain a significant amount of relevant study data. Therefore, a common approach in software engineering studies is the recruitment of convenience samples, such as computer science (CS) students (e.g., [107–111]). While recent work indicates that CS students can be acceptable proxies for professionals in security studies [43, 49, 54, 112], there are still some caveats [54]. Similarly to end users, software developers usually consider security a secondary task [2, 105]. Thus, it is unclear whether a developer usability study that specifically provides security instructions can adequately reflect real-world circumstances. More research is needed to provide context for recruitment strategies and to examine research methods for exploring developers' security behavior. In addition to the primary-level analysis mentioned above, this work contributes to *meta-level* knowledge of methods for conducting security-related studies with developers.

1.2. Contribution

In order to offer deeper insights into the research questions addressed in this thesis, I designed a usability security developer study that considers several aspects [113]. Participants were asked to develop a user registration function for a university social networking platform. In order to examine whether software developers think about security on their own, half the participants were explicitly *prompted* to store user passwords securely, while the other half were told that the study was about API usability (*non-prompted*). The study also investigated whether an API that offered inbuilt functionalities for secure password storage would help developers produce more secure code than an API that required developers to manually choose secure password storage mechanisms. Based on their popularity, Spring [114] and JavaServer Faces (JSF) [115] were chosen as frameworks for the study. Spring was chosen as a representative of a more supportive framework with inbuilt functionalities for secure password storage. In contrast, with JSF, developers must implement secure password storage on their own. Due to the limited amount of research experience with the design of usability security studies with developers, a qualitative study in a laboratory setting was chosen for the initial approach. As mentioned above, recruiting professional software developers is rather challenging. Therefore, 20 CS students at the University of Bonn were recruited for the study. In this study, none of the participants stored user passwords securely without being prompted. To further explore the issues and methodological requirements, the study was extended to include an additional 20 CS students, and a quantitative analysis was performed [116]. The quantitative analysis confirmed the results of the qualitative study.

However, in the qualitative and quantitative studies with CS students, some participants claimed they would have behaved differently if they were carrying out the task for a real company. In order to explore whether the previous results were a study artifact due to the lab and study setup, a follow-up study was conducted. Due to difficulty of recruiting professional developers from organizations, freelance developers were recruited online via Freelancer.com [117]. This time, participants were hired to do a programming task. The pilot study revealed that the university context might have led participants to believe they were solving university homework tasks. Therefore, the study context was changed from university researchers to start-up employers. After completing the programming task, participants were informed about the research context of the study and were asked to fill out a survey. In the freelancer study, participants' behavior was similar to that of CS students with regard to secure password storage practices.

Finally, one open question remained: Do professional developers employed by companies behave the same as CS students and freelance developers? To address this research question, an additional online study was conducted with professional developers from different companies [118]. This study used the same design as the previous freelancer study [117]. However, it was not realistic to hire professionals from companies for a small task without revealing the research context of the study. Therefore, participants were informed that the programming task was requested for research purposes. In this study, professional developers employed in companies chose higher security implementations for password storage than CS students or freelancers. However, similar to CS students and freelancers their security behavior varied when they were not prompted for security. In addition, like the students, professional developers working in companies implemented higher security standards when using Spring rather than JSF.

In the primary analysis, the following findings were observed in the four studies:

- **CS students' password-storage practices**: CS students had misconceptions about algorithm and parameter choices for secure password storage. Thus, they often chose poor mechanisms for secure password storage.

- **Freelance developers' password-storage practices**: Freelancers had similar misconceptions to those of CS students about algorithm and parameter choices for secure password storage. Thus, they often chose poor, but also inappropriate mechanisms for secure password storage.

- **Company developers' password-storage practices**: Professional developers employed in companies performed better than CS students and freelancers with regard to algorithm and parameter choices for secure password storage. Thus, they often chose strong, appropriate mechanisms for secure password storage.
- **Framework support**: There are early indications that a framework that offers inbuilt functionalities for secure password storage can help developers produce more secure code than an API that requires developers to manually choose secure password storage mechanisms.[1]

The meta-level analysis revealed additional results:

- **Security prompting**: Security prompting had a significant effect on participants' behavior regarding secure password storage. This effect was found for CS students, freelancers and professional developers employed in companies.
- **Qualitative vs. quantitative studies**: There are early indications that qualitative studies with developers can offer valuable results without the need to conduct quantitative studies to confirm the findings.
- **CS students in security developer studies**: Professional developers employed in companies performed better than CS students in absolute terms, that is, they made better decisions with regard to secure password storage. However, there are early indications that, in relative terms, the findings of security developer studies conducted with CS students are comparable to those of studies conducted with professional developers employed in companies. An example of a relative finding is that both participant groups achieved higher security with API A (Spring) than with API B (JSF). Freelancers' results were similar to those of students in relative, but also in absolute terms.

1.3. Thesis Structure

The findings outlined above have been published in four peer-reviewed, high-quality conference research papers of which I am the first author. In this thesis, each paper is summarized in one chapter; the chapters are ordered in order of publication (Chapters 4 to 7). To provide context, the study design and evaluation are presented in Chapter 2 and previous related work in Chapter 3. Thus, the thesis is structured as follows:

[1]This was tested with CS students and developers employed by companies but not with freelance developers.

Chapter 1: This introductory chapter describes the research questions addressed in the thesis, the contribution of these findings to existing knowledge, and the structure of the thesis.

Chapter 2: This chapter describes the study design and evaluation methods. Most of the content of this chapter was previously published in the papers [113, 116] described in Chapters 4 and 5.

Chapter 3: This chapter summarizes previous work on password storage, API usability, ecological validity, and developer security behavior. Much of the content of this chapter was previously published in the papers [113, 116–118] described in Chapters 4 to 7.

Chapter 4: This chapter describes a qualitative password-storage study conducted with CS students. The contents of this chapter have been published as part of the paper: "Why Do Developers Get Password Storage Wrong? A Qualitative Usability Study." Alena Naiakshina, Anastasia Danilova, Christian Tiefenau, Marco Herzog, Sergej Dechand, Matthew Smith. *In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (CCS 2017) [113].

Chapter 5: This chapter describes a quantitative password-storage study conducted with CS students. The contents of this chapter have been published as part of the paper: "Deception Task Design in Developer Password Studies: Exploring a Student Sample." Alena Naiakshina, Anastasia Danilova, Christian Tiefenau, Matthew Smith. *Fourteenth Symposium on Usable Privacy and Security* (SOUPS 2018) [116].

Chapter 6: This chapter describes a field study on password-storage conducted with freelance developers. The contents of this chapter have been published as part of the paper: "'If you want, I can store the encrypted password.' A Password-Storage Field Study with Freelance Developers." Alena Naiakshina, Anastasia Danilova, Eva Gerlitz, Emanuel von Zezschwitz, Matthew Smith. *In Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (CHI 2019) [117].

Chapter 7: This chapter describes an online study on password-storage conducted with software developers employed by companies. The contents of this chapter have been published as part of the paper: "On Conducting Security Developer Studies with CS Students: Examining a Password-Storage Study with CS Students, Freelancers, and Company Developers." Alena Naiakshina, Anastasia Danilova, Eva Gerlitz, Matthew Smith. *In Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (CHI 2020) [118].

Chapter 8: This chapter summarizes the results of the studies described in Chapters 4 to 7. Based on the findings, it outlines the impact of software developers' security behavior on software security and the methodological implications of study and task design on security developer studies. In addition to that, recommendations for future research were provided.

Chapter 9: This chapter summarizes the findings of this thesis and outlines suggestions for future work.

Disclaimers at the beginnings of Chapters 4 to 7 summarize my co-authors' and my contributions to the work. The plural personal pronoun "we" will be used in place of "I" to demonstrate that I received much-appreciated help from colleagues and co-authors with the studies described in this thesis. Without this help, much of this work would not have been possible. Therefore, I would like to thank them for their incredible support.

2. Background

This chapter will describe the study which is the baseline for the research presented in this thesis. First, Section 2.1 introduces the study design, including the programming task and the deception used. Second, Section 2.2 describes the code evaluation technique, including the security scale. Third, Section 2.3 summarizes the hypotheses and statistical tests used for the statistical analysis.

2.1. Study Design

In order to gain insights into the development process of password storage and user authentication code, I designed a usability study for developers by considering two important aspects. First, the effect of telling participants to store the end-user passwords securely as opposed to think of security on their own. Second, the effect of giving developers a framework which offers inbuilt functionality for securely storing passwords. Such a framework would remove the burden on the developer of having to choose a secure salt and hash function, as well as knowing that iterations are recommended [97].

As is common practice in usability studies, I created an authentic scenario to give the participants context. Participants were asked to create code for user registration and user authentication of a social networking platform. I selected two different Web frameworks with different levels of support for secure password storage and divided the participants between these two frameworks. I refer to [119], where a taxonomy of categorization of levels that support a framework is explained in further detail. I chose frameworks with two levels of support for the study: (1) *manual*, the weakest level of support in which developers have to write their own salting and hashing code using just crypto primitives, and (2) *opt-in*, a framework which offers a secure implementation option, which can be used by developers if they think of it or find it. I did not find any framework which offers the level of *opt-out* support for password storage.[1]

Furthermore, half the participants received an explicit request for secure password storage and the other half were subjected to a deception task. Only after completing the study, participants were informed about the true purpose of the study. This deception was necessary since requesting all participants for secure password storage would have influence those participants who had not been explicitly told to securely store the passwords, thus making it impossible to discover whether they had thought of secure storage on their own.

[1]This is probably due to the fact that it would be very hard to stop a developer from storing arbitrary strings in plain text. The study results show, however, that this would be a very desirable level of support to reach.

Condition	Framework	Level of Support	(Non-)Prompting
1	JSF	manual	Prompting
2	JSF	manual	Non-Prompting
3	Spring	opt-in	Prompting
4	Spring	opt-in	Non-Prompting

Table 2.1.: Study conditions

In Table 2.1 an overview of the resulting study conditions is available. In the following sections, the rationales behind the choices made in the study design will be discussed.

2.1.1. Language and API Selection

Several statistics based on diverse analytical data indicate Java as one of the most widely employed server-side programming languages for applications and in the Web [120–125]. Java is also the main programming language taught at the University of Bonn[2]; thus Java was chosen as programming language for the study.

To compare the different levels of support offered by frameworks and their APIs, I conducted an analysis of the most popular Web frameworks based on the data gathered from the open source platform HotFrameworks [126]. HotFrameworks combines scores from GitHub [127] (number of stars a git repository has for a framework) and Stack Overflow [128] (questions tagged with a name of a framework) and creates a statistic of the most popular Web frameworks. Additionally, I used Google Trends [129] in order to verify the data offered by HotFrameworks. Google Trends provides a good overview regarding the development of interest in frameworks worldwide. I concentrated on frameworks that have been popular for the past five years. I selected *Spring* and *Java Server Faces (JSF)* as the most suitable Web frameworks for the study in terms of popularity and level of support for secure end-user password storage as explained above. In order to store a password securely with Spring, the developer can make use of Spring Security's `PasswordEncoder` interface. By contrast, participants implementing an application with JSF have to manually implement secure password storage or decide to integrate a library on their own.

2.1.2. Programming Task

The aim was to was to design a Web development task that was short enough to be solved in one working day but also long enough to hide the fact that secure password storage was the only thing of interest. It should also be a longer task than the recent API comparisons of Acar et al. [50] to observe participants having to potentially prioritize functionality or security. This is similar to the

[2]At the time of the study design in 2016/2017.

real world, where functionality of an application is the primary task and security is the secondary task [4, 130, 131].

Bonneau and Preibusch defined three broad categories of websites: identity sites (e.g. social network), e-commerce sites and content sites (e.g. newspaper websites) [98]. On the one hand, I wanted to avoid exaggerate awareness of security arising out of e-commerce sites. On the other hand, content sites offer little incentive to implement security. Therefore, I took an identity site as scenario for the study: a social networking platform.

Four variants on how the study was conducted were examined (see Table 2.1). I was interested in the qualitative feedback on Spring and JSF as well as when students were directly asked to implement secure password storage (henceforth called the prompted group) and when they were not prompted (henceforth called the non-prompted group).

The application which the participants were supposed to extend was designed with the standard Model-View-Controller pattern. The front end was provided and the database back end fully implemented and documented. To ease the integration, students were given an illustration of the layers for the whole application and documentation of the existing Java classes with implementation hints (see Appendix A.6, A.7). To facilitate a natural process, participants were explicitly allowed to use any kind of information source that could be found on the Internet.

2.1.3. Deception and Security Prompting

Interactions with participants in research studies must align with ethical considerations. Using *deceptive techniques* and misleading participants can reduce their trust in the research, raising suspicions towards investigators and future studies. Deception can even reduce public trust in scientists and thereby harm society at large [132]. However, in research on certain areas of human behavior, some deception may be necessary in order to obtain ecologically valid results. That is, sometimes participants must be deceived for researchers to observe natural, real-world behaviors. Consequently, a more realistic experimental situation enhances a study's internal validity [132].

The use of *priming* can be one form of deception. The concept of priming originates in social psychology; the term was first used by Karl Lashley [133] in 1951. He examined the "logical and orderly arrangement of thought and action" [133, p. 112]. Lashley associated priming with "patterns of subthreshold facilitation pervading the network of neurons which is activated by the more specific external stimulus" [133, p. 134]. In other words, priming is "a mechanism to increase the probability of a behavioral response" [134, p. 209]. Nearly 50 years later, in 2000, Bargh et al. [135] pointed out that this may be done subconsciously: "Priming – how recent or current experience *passively* (without an intervening act of will) creates internal readinesses" [135, p. 3]. In a meta-analysis, Weingarten et al. [136] investigated the effects of incidentally introduced words on behavior (e.g., achievement words such as win, compete, succeed, strive, attain, achieve, or master) and found that researchers have significant flexibility in the selection of priming study parameters. Introducing

Publication	Priming	Non-Priming
Schechter et al. [137]	"[Participants] played a role, told that their role was concerned about security."	"[Participants] played a role, given no indication that security is focus of study."
Kelley et al. [138, p. 526]	"[...] the email scenario was designed to elicit higher-value passwords."	"The survey scenario was designed to simulate a scenario in which users create low-value passwords [...]."
Fahl et al. [71, p. 4]	"We also asked the participants to act as if the passwords for the fictitious study scenario were real passwords."	"The word "password" was not used at all."

Table 2.2.: Security Priming in end-user password studies

Publication	Prompting
Bau et al. [139]	"To remind the freelancers about security, we mentioned that our site was mandated by "legal regulation" to be "secure", due to hosting photos of minors as well as storing sensitive contact information. We explicitly mentioned only those logged in as a member of a team (as admin, coach, or parent) should be able to access any of the sensitive photos, files, or contact information. We asked the freelancers to enforce these policies but did not offer any further guidance on implementation. We also did not provide any explicit security specifications outside of those mentioned above."

Table 2.3.: Security Prompting in a developer password study

incentives or personal values ensures a high value for the primed behavior or goal.

In the field of computer science, priming was investigated in password studies conducted rather with end users [71, 137, 138]. Table 2.2 shows an overview of previous studies and the priming variable used in each. In research on website authentication protection mechanisms, Schechter et al. [137] explored deception in the form of a priming effect. This study included three groups of end users. Participants in the first two groups were asked to role-play working with authentication data in a bank setting. One group was primed; they were told that their character in the role-play was concerned about the security of the password. In another study on the strength of text-based passwords, Kelley et al. [138] used a form of priming without using the term "priming." Participants were advised to create a password for either a survey or an email account. Since emails are associated with more sensitive data than surveys, the authors predicted that participants would choose stronger passwords for the email account than for the survey. Finally, Fahl et al. [71] conducted a between-groups study with two variables (lab vs. online study, priming vs. non-priming). They compared real-world password choices with passwords chosen by end users in a study environment in primed and non-primed conditions. The primed group was asked to create and manage passwords as they would in real life; for the unprimed group, the term "password" was not mentioned in the introductory text at all. Neither Schechter et al. nor Fahl et al. found that priming had a significant effect. Kelley et al., however, observed that users chose stronger passwords in a hypothetical email scenario than in a hypothetical survey scenario.

The studies described above were conducted with end users, though previous research has shown that the mental models and security behavior of experts, such as developers, differ from those of end users [9, 140, 141]. Therefore, these results might not apply to software developers. In a technical report, Bau et al. [139] asked freelance developers to create a social networking web application requiring users to register. In the task description, participants were reminded that the website needed to be secure because it would store sensitive information about registered users (see Table 2.3). Although the authors did not specify the steps to ensure security, the chosen wording provided a "security prompt" rather than "priming." Weingarten et al. [136] differentiate between priming and "direct goal induction" [136, p. 477]. According to Weingarten et al., in priming, "a goal [is] primed incidentally, without calling attention to the connection between the priming task and the outcome task or trying to induce intentional behavior" [136, p. 477]. In contrast, "explicitly assigned goals in the form of an instructional set" [136, p. 477] are classified as direct goal inductions. Therefore, if priming is associated with subconscious stimulus, a security demand for the application can be classified as security prompting. In Bau et al. [139], six of the eight participants at least hashed the user passwords; the other two stored user passwords in plain text. In that study, participants also completed a security quiz with a question about password-storage security ("*What standard security measures should be taken when storing user passwords to a database?*" [139]) either before or after they worked on the programming task: "*For the freelancers, we administered the quiz well before or after the development cycle [...]*" [139]. Thus, participants might have been primed for secure password storage, resulting in better password-storage security than would have been implemented without priming.

In order to provide deeper insights into this field and the ecological validity of security developer studies, the present study was designed to investigate whether concealing or sharing the true purpose of a study affects participants' behavior with regard to secure storage of user passwords. To examine this more explicitly than Bau et al. [139] (see Table 2.3), half of the participants were *prompted* to focus on security through the direct instruction: "*Please ensure that the user password is stored securely*." The other half were deceived and told the study was about API usability.

Table 2.4 shows three stages at which deception can be integrated [142–144]: subject recruitment, research procedure and post-research/application. This study investigates whether developers make sure to store end users' passwords securely if they are not explicitly told to do so. To avoid priming participants with the use of the term "security," the recruitment text only contains information about an API usability study in Java; it makes no mention of security at all. Consequently, deception was used in the subject-recruitment (*purpose of research*) phase for all participants. The study invitation can be found in Appendix A.1. Additionally, only the security-prompted participants were informed of the study's purpose in the task description. Thus, non-prompted participants were deceived during the research procedure through *misrepresentation of the study's purpose* and *withholding of information*. While researchers should always weigh all options before using deception in research, Kimmel [144] points out that deception can be "positive (i.e., beneficial to recipients) or negative (i.e., harmful to recipients); [...] short- or long-term and immediate or delayed" [144, p. 667].

In order to minimize potential harm to participants' perceptions, attitudes, behaviors, or belief in scientific research, it is essential to debrief all participants at the end of the study. In view of the lack of existing knowledge on methodological' approaches to security developer studies and

Subject Recruitment	**Research Procedure**	**Postresearch/Application**
Identity of researcher and/or sponsor	Misrepresentation of purpose	Violation of promise of anonymity
Purpose of research	False information about procedures, measures, etc.	Breach of confidentiality
Participation incentives	Withholding information	Misrepresenting implications of research results
Involving people in research without their knowledge	Concealed observation	False feedback during debriefing session

Table 2.4.: Use of deception at various stages of the research process.
Source: Table originally from Kimmel in [142, p. 53]

the massive threat to sensitive user data posed by developers' use of insecure mechanisms to store user passwords, I believe the present study is important and deception is unavoidable. Costs to participants might include embarrassment in front of the researcher; however, participants might also benefit from increased self-insight and an increase in valuable programming knowledge about security. Additionally, the increased degree of methodological control provided by deception and the potential for improved understanding of developers' security behavior as well as the possibility to validate theories, previous research, and assessment instruments outweigh any potential costs [144].

2.2. Code Evaluation

For evaluation, the functionality and security of the participant's program code was examined. If the end user was able to register to the Web application, meaning that her/his data provided through the interface was stored to a database, then the task was solved functionally to an adequate degree. In order to store end-user passwords securely, participants had to hash and salt them [97]. Two measures were used to record the security of a participant's solution. Every solution was rated on a scale from 0 to 7, according to the security scale introduced in Section 2.2.1. This value is referred to as the *security scale*. In addition, a binary variable called *secure* which was considered if participants used at least a hash function in their solutions and thus did not store the passwords in plain text.

2.2.1. Security Scale

Nowadays, GPUs can run billions of instructions per second [145]. Consequently, the one-way hash functions `SHA1` [146, 147] and `MD5` [148] can be calculated extremely quickly. Thus, these algorithms are no longer considered as sufficiently secure in most cases, as demonstrated by several serious password leaks within major organizations [95, 102, 103]. To counter this problem, more computationally intensive password hashing schemes (PHSs), like `PBKDF2 (Password Based Key Derivation Function)` [149], `bcrypt` [150] and `scrypt` [151] have been proposed. These algorithms apply the idea of *key stretching* to increase the computation time for testing each possible password (*key*) by iterating the hash of the salted password [152]. Due to the presence of cheap and parallel hardware, the widely used `PBKDF2` and `bcrypt` are vulnerable to massively-parallel attacks [145, 153, 154]. In order to counter these attacks, memory-hard PHSs were introduced, since fast memory is considered expensive. `Scrypt` is a memory-hard PHS, but it is vulnerable to other types of attacks such as cache-timing and garbage-collector attacks [155, 156]. By contrast `Argon2` [157], the winner of the Password Hashing Competition (PHC) [158], provides a defense against such attacks and can be viewed as the state of the art in the design of memory-hard functions. Other promising PHSs can be found in [153, 159].

Although we are aware of the fact that memory-hard functions like `scrypt` and `Argon2` are stronger than `bcrypt` and `PBKDF2`, we decided not to penalize participants for using current standards [97]. While NIST indicates the use of a suitable one-way key derivation function for password storage as a requirement to be followed strictly, the usage of memory-hard functions is recommended as particularly suitable. As an example for a suitable key derivation function `PBKDF2` is still suggested with an iteration count of at least 10 000. In previous specifications [160], NIST even recommended an iteration count of at least 1 000. Since the study with CS students was conducted in May 2017 [113], only a draft [161] of NIST's current specification [97] was available to the participants. Therefore, a solution with at least 1 000 iterations was accepted; however bonus points were awarded if participants changed the iteration count to at least 10 000.

In order to store a password securely with Spring, developers can make use of Spring Security's cryptography module `spring- security-crypto`[3], which contains its own password encoder. Developers can simply import an implementation of the password encoder with a hash function they prefer. Although an implementation of `scrypt` is available, the official Spring Security Reference [162] refers to `bcrypt` as the preferred hash function at the time of this study design. The provided implementation uses a cost parameter of 10 resulting in $2^{10} = 1\,024$ key expansion iterations by default. Originally Provos and Mazieres proposed a cost parameter of 6 for end-user and 8 for administrator passwords [150]. While these values are considered outdated today, recent works use the cost parameter 12 for `bcrypt` as a common choice for analysis [153, 163–165]. By using Spring Security's `bcrypt` implementation, a random salt value of 16 bytes is integrated by default as well. Since it was not the aim of the study to penalize participants for using standards of frameworks, parameter defaults of Spring Security's `bcrypt` implementation were rated as secure.

By contrast, an application implemented solely with JSF can either be extended by Spring Security, or the developer can implement her/his own solution for secure password storage. For instance, she/he could make use of Java's Cryptography Extensions (JCEs). A random salt value has to be implemented by the developer as well. Participants were not restricted in their choice of how to implement secure password storage, neither for Spring nor for JSF.

To summarize, security of end-user password storage was rated under different assessment criteria. The highest possible score is 7 points, consisting as follows [97, 166]:

1. The end-user password is salted (+1) and hashed (+1).

[3]Since Spring Security 3.1.

2. The derived length of the hash is at least 160 bits long (+1).
3. The iteration count for key stretching is at least 1 000 (+0.5) or 10 000 (+1) for `PBKDF2` and at least $2^{10} = 1\,024$ for `bcrypt` (+1).
4. A memory-hard hashing function is used (+1).
5. The salt value is generated randomly (+1).
6. The salt is at least 32 bits in length (+1).

As explained above, in the following analysis, solutions which achieved 6 *or* 7 points are referred to as secure solutions according to the security scale. Solutions which earned 6 points followed industry best practices. In other words, from a usability perspective, these participants did all one could realistically expect from them.

2.3. Statistical Analysis

2.3.1. Hypotheses & Tests

Seven main hypotheses were examined for the quantitative analysis. Two concerned the meta-focus, namely, the effect of prompting/deception, denoted by P(rompting). Two further concerned the A/B test comparing the two frameworks, denoted by F(ramework), and the final three were general tests concerning password storage security, denoted by G(eneral).

H-P1 Prompting has an effect on the likelihood of participants attempting security.

H-P2 Prompting does not have an effect on achieving a secure solution once the attempt is made.

H-F1 Framework has an effect on the security score of participants attempting security.

H-F2 Framework has an effect on the likelihood of achieving functional solutions.

H-G1 Years of Java experience have an effect on the security scores.

H-G2 If participants state that they have previously stored passwords, it affects the likelihood that they store them securely.

H-G3 Copying/pasting has an effect on the security score.

2.3.2. Meta-study

It is natural to assume that requesting a secure solution will lead to more attempts at security (H-P1). The interesting aspect here was how many of the non-prompted participants attempted a secure solution. While prompting was expected to increase the number of attempts, it was not expected to receive a different failure rate between the prompting and non-prompting group (H-P2), i.e., if non-prompted participants attempted security, they should not have failed more often than prompted members.

2.3.3. Primary Study

While only indirectly linked to security, the possibility that the differences between the two frameworks (JSF and Spring) could lead to different rates of functionality (H-F2) was considered. It was also expected that the greater level of support offered by Spring would increase the security score of Spring participants (H-F1).

2.3.4. General

The above hypotheses are novel to this work. However, it was also desirable to confirm findings from related studies. In their study, Acar et al. [49] observed that the programming language experience had an effect on the security of participants' solutions. Therefore, it was assumed to observe an effect regarding experience with the Java programming language and the security score of participants' solutions (H-G1). In addition, it was assumed that if participants had experience with storing user passwords in a database backend, they would be more likely to create a secure solution in the study (H-G2). Finally, several studies noted the effects of copy/paste on study results [50], especially in terms of security [51, 112, 167, 168]. Thus, it was assumed that copy/paste events would affect the security code of the participants as well (H-G3).

2.3.5. Statistical Testing

For statistical analysis the common significance level of $\alpha = 0.05$ was chosen. When conducting tests on all participants, the groups were labeled as "all". When only testing subgroups, these were also labeled for easy interpretation. Fisher's Exact Test (FET) was used for categorical data. Linear regression was considered for numeric data and logistic regression for binary data if the data was normally distributed. In order to test for normality, the Kolmogorov-Smirnov test was used. Additionally, the data was plotted for manual inspection. For data that was not normally distributed, the following non-parametric tests were used: in order to find differences between all four conditions, the Kruskal-Wallis test was used; for two groups, the Mann-Whitney U test was used. Statistically significant values are indicated with an asterisk (*).

Of the seven main hypotheses, three concerned the security score, two concerned the binary secure value, one concerned the attempted security, and one concerned functionality. Since all but the functionality tests were closely related, a family-wise error correction using the Bonferroni-Holm method with a family-wise correction of 6 was applied. However, in the exploratory part of the analysis, multiple testing correction was not applied. Since virtually no research has been conducted on study design for developer studies, it was more important to discover interesting effects for future research to explore than it was to avoid type 1 errors while potentially dismissing an important effect merely because the sample size was not big enough (i.e., type 2 errors). For more information on family-wise errors, see [169]. To ease identification, Bonferroni-Holm corrected tests were labelled with "family = N", where N was the family size. Both the initial and corrected p-values are reported.

3. Related Work

Compared to the amount of usability security research focused on end users (e.g., [5–40]), only a handful of usability security studies have been conducted with administrators (e.g., [20, 41–47]) or software developers (e.g., [48–54]). This chapter covers previous literature on security software development and user studies with experts and is divided into five parts. First, related work in the area of password storage security is discussed in Section 3.1. This section is further divided into two subsections: a technical analysis of password storage and previous studies on password storage conducted with developers. Second, Section 3.2 discusses work on the usability of crypto APIs. Third, the ecological validity of user studies is discussed in Section 3.3. Fourth, previous developer studies addressing security beyond the examination of password-storage or crypto API usability are presented in Section 3.4. This section is also further divided. Section 3.4.1 focuses on developer studies conducted with student participants. Section 3.4.2 discusses developer studies with freelancers, and Section 3.4.3 summarizes papers comparing previous studies with professionals and students. Finally, Section 3.5 describes the contributions of the present study to gaps in previous research.

3.1. Studies on Password-Storage Security

Passwords are a famous subject in the field of usable security research and many studies have been conducted to find out why end users create passwords the way they do or how they could be influenced in order to choose more secure passwords [55–93, 170]. While it is important to find secure and usable password solutions for end users, it is also essential to understand how developers manage end-user password storage. 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.

3.1.1. Passwords - Technical Analysis

Bonneau and Preibusch [98] carried out an empirical analysis of password implementations deployed on the Internet. They evaluated 150 websites which offered free user accounts and relied on user-chosen textual passwords for authentication. All of the sites showed inconsistent implementation choices impacting security. For instance, e-commerce sites deployed TLS for password transfer and notification emails in password reset cases, whereas identity providers protected against weak passwords and guessing attacks. Thereof, they deduced different threat models: while identity sites worry about guessing attacks by casual acquaintances, e-commerce providers fear hacking by profit-seeking criminals. Still, Bonneau and Preibusch found evidence that websites storing payment details as well as user data provided appropriate password security. By contrast, the worst security practices were found on content websites having few security incentives. Most of them suffered from a lack of encryption to protect transmitted passwords. Further, storage of plaintext passwords in server databases with little protection of passwords against brute force attacks were commonly-applied practices. Bonneau and Preibusch concluded that these sites deploy passwords primarily for psychological reasons, both as a justification for collecting marketing data and as a way to build trusted relationships with customers.

Finifter and Wagner [119] conducted an evaluation of the code produced in a Web development competition of Prechelt [52]. They searched for correlations between programming languages/framework support for security aspects and the number of vulnerabilities. They did not find a relationship between a programming language and application security. However, automatic features were effective at preventing vulnerabilities in general, but not for secure password storage. Although many of the used frameworks provided “automatic” support for storing end-users’ passwords securely, the developers did not employ them. A finding which motivated the study presented in this thesis to conduct a deeper examination of frameworks with different password-storage support.

Florêncio et al. examined the security of web services, considering password implementation choices, policy, and administration. This study includes a literature review and uses “first-principles reasoning” [95]. The authors characterized users’ web services accounts, online attacks, and offline attacks to conduct a systematic analysis. While online attacks on passwords occur at the front end of web services, offline attacks are executed at the server-side/back end. According to Florêncio et al., attackers probably need to make more than 10^6 guesses for online attacks and more than 10^{14} for offline attacks to succeed. This illustrates the enormous gap in the effort needed to withstand online and offline attacks, especially since offline attacks are likely much rarer than generally believed. Attackers need to perform offline attacks beyond rainbow tables if they obtain access to a database with passwords that are stored as salted hashes. However, recent large-scale breaches have demonstrated that passwords are rarely salted and hashed. The effort demanded of end users when complex passwords are required is wasted if software developers store user passwords in plaintext in databases or use inappropriate techniques such as reversible encryption for password storage.

The findings of the research presented above motivated the present study, which seeks to gain deeper insights about *why* developers fail to securely store user passwords in databases.

3.1.2. Passwords - Developer Studies

Balebako et al. [171] interviewed 13 app developers and surveyed 228 to learn how they make decisions about privacy and security. This study found that developers in smaller companies often rely on recommendations from friends, social networks, and search engines like Google, while bigger companies often employ security experts, resulting in better privacy and security behaviors. Half of the developers surveyed reported that they securely store data in databases. However, this assertion should be eyed with caution considering the findings presented in Section 3.1.1 and in the following work.

Acar et al. [49] conducted a security developer study with 307 active GitHub users, where participants were requested to implement three security related tasks (URL shortener, credential storage, and string encryption). The authors also asked them to complete a questionnaire about their programming experience, security background, and occupation. A number of participants stored user passwords in plaintext (in 46.9% cases), did not use a proper salt (in 77.1% cases), about 14% used unsafe hash functions like MD5 or SHA-1, and 4.2% used the discouraged practice of encryption for password storage. Neither the self-reported status as student or professional developer, nor the security background of the participants correlated with the functionality or security of their solutions. However, the authors found a significant effect for Python experience on functionality and security of program code.

Bau et al. [139] explored Web application vulnerabilities. They used a metric to search for correlations between the vulnerability rate of a Web application and (1) the nature of a developer team (start-up company vs. freelancers), (2) security background knowledge of developers, and (3) programming languages. For the start-up group, existing programs were analyzed. For the freelance group, eight compensated developers were invited to participate in a developer study. As compared to the start-up group, the freelancers were prompted for security in the task description. They found that Web applications implemented with PHP or by freelancers showed higher rates of vulnerabilities. With regard to user password-storage security, Bau et al.'s study results showed that of the eight freelancers, three participants stored a salted hash, another three participants stored only a hash without salting the password before, and the remaining two participants stored the user passwords in plaintext. Considering a security quiz participants had to answer before or after completing the programming task, the authors observed discrepancies between the expertise and the resulting implementation solutions of freelancers. The authors concluded that further research is required to better understand developers and their practices. As with the previous study, this study only looked at the code and said nothing about the thoughts of the developers. Therefore, the study presented in this thesis provide deeper insights into developers' thoughts about secure password-storage implementation. In addition to that, prompting developers is analyzed as a variable.

Nadi et al. [104] studied the kinds of problems developers struggle with when using APIs. They analyzed the top 100 cryptographic questions on Stack Overflow as well as 100 randomly selected

GitHub repositories that used Java crypto APIs. Within the analyzed projects, they found passwords being encrypted, a discouraged practice, which should be replaced by hashing. Afterwards, they conducted a study with 11 developers and a survey with 48 developers. They found poor documentation, lack of cryptography knowledge, and bad API design as the main obstacles developers struggle against. Code templates, tools to catch common mistakes and better documentation that includes examples were suggested for solving problems.

One related usability study on password storage was conducted by Wijayarathna and Arachchilage [172] with 10 GitHub developers. The authors evaluated the usability of Bouncycastle API's functionality of the password hash function `scrypt`. Participants worked on their own computers and had to improve the password storage of a simple web application by using `scrypt`. On observing 63 usability issues of the `scrypt` implementation based on the Bouncycastle API, the authors found that most of the difficulties arose from the complexity of the method's parameters. The authors concluded that the usability and security of APIs should be improved to increase software quality. In accordance with the study findings of this thesis, Wijayarathna and Arachchilage argued that developers usually are not security experts and thus they should be provided appropriate guidelines, especially in organizational scenarios.

3.2. Usability of Crypto APIs

Software developers often use crypto APIs to protect the sensitive data processed by current applications. However, due to their complexity, developers often struggle to use them correctly [104, 172–174]. For example, Lazar et al. [174] found that 80% of the analyzed Common Vulnerabilities and Exposures (CVEs) resulted from misuse of cryptographic APIs by developers. Additionally, Egele et al. [173] investigated the misuse of cryptographic APIs for Android application developers. Of the more than 11 000 Android apps analyzed, 88% featured at least one of six cryptographic errors.

Green and Smith [2] point out that "normal" developers are usually not security experts and that designers of crypto APIs must keep this in mind. Green and Smith argue that "developer-friendly and developer-centric" [2, p. 1] tools and APIs are required for developers to carry out complex security tasks. In [2], Green and Smith discussed the issues of crypto APIs lacking usability and provided security relevant examples, such as insecure password storage. Finally, the authors proposed ten principals for designing usable, secure crypto APIs that could reduce developer errors. The first and most important of these is to integrate crypto functionality into standard APIs so that developers do not need to interact with crypto libraries at all. For example, safe defaults for secure password storage should be provided so that developers are not expected to make security decisions such as which hash function to use, how to generate the salt value, or how many iterations are needed. Usability characteristics relevant for security APIs were also discussed by Gorski et al. in [175], whose analyses was based on studies investigating API usability. For instance, they referred to a user study of Fahl et al. [51]. Fahl et al. conducted qualitative interviews with software developers who deployed malfunctioning Secure Sockets Layer (SSL) issues revealing usability problems in the API design. They presented a framework in order to help developers implement secure software.

Moreover, Stylos and Myers [176] conducted studies with developers to examine API design usability with regard to method placement. They created two different versions of three different APIs and asked ten developers to implement three small tasks. To capture the participants' expectations, the researchers first let them write pseudocode for how they would expect to solve the task, before looking at any real API. They found that developers solved a task faster when using APIs in which the classes from which they started their exploration included references to the other classes they needed. In [177], Myers and Stylos also 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.

Acar et al. [50] conducted a quantitative evaluation of the usability of several crypto APIs. They conducted an online study with open source Python developers recruited from GitHub to test the following five Python crypto libraries: cryptography.io, Keyczar, PyNaCl, M2Crypto, and PyCrypto. They tested symmetric and asymmetric encryption and key generation as well as certificate validation. Their tasks were designed to be short enough so that the uncompensated participants would be likely to complete them before losing interest, but still complex enough to enable errors to be made. The study results revealed that even for simplified libraries, security success was under 80% in most cases. Furthermore, 20% of the functional solutions were incorrectly rated by participants as being secure. The authors concluded that, while cryptographic libraries should offer a simple and convenient interface, this is not enough. Developers also need support with a range of common tasks and should be provided accessible documentations with secure, easy-to-use code examples. However, due to the short task duration and online nature of the study, the authors could not do an analysis of the causes. The qualitative nature of the study presented in this thesis, in combination with the much longer task, allows to give a deeper analysis.

Overall, in contrast to the previous studies, the studies presented in this thesis are examining password storage and whether prompting participants regarding security in the task description encourages the creation of secure code. Additionally, the usability of APIs with different levels of password-storage support is examined. In addition to quantitative, qualitative methods are used to gain deeper insights into the rationale behind developers' security behavior.

3.3. Ecological Validity

As highlighted by Acar et al. [105], the examination of ecological validity–whether or not a study reflects the real world–is of high importance, especially in the area of security studies conducted with the specific group of software developers.

Several previous studies conducted with end users examined whether participants' study results reflected peoples' real-world behavior. For example, the studies of Schechter et al. [137] and Fahl et al. [71], which were already described in Section 2.1.3 in more detail. Schechter et al.'s [137] study of website authentication protection mechanisms used the idea of *priming*. The authors conducted a study with three groups. The participants of the first two groups were asked to role-play working with authentication data in a bank setting. One of these groups was thereby primed for security. Further, Fahl et al. [71] conducted a between-groups study with two variables (lab vs. online study, priming vs. non-priming). The authors compared real-world password choices with passwords chosen by end users in a study environment, considering priming and non-priming conditions. While the primed group was asked to behave as in real life when creating and managing passwords, in the introductory text of the non-primed group the term "password" was not mentioned at all. Neither Schechter et al. nor Fahl et al. found a significant effect of priming.

Mazurek et al. [57] investigated the guessability of university users' passwords. In this study, the login passwords of university faculty, staff, and students were attacked with a state-of-the-art password cracking algorithm. The set of university passwords was also compared to passwords from previous studies and cracked leaked passwords. The authors found that by considering similar composition policies, university users' passwords were more similar to passwords collected for research than to those from leaked databases.

Wash et al. [89] examined whether end-users' self-reported password behavior reflects their actual password behavior. In this study, password entry events of 134 participants' web browsers were analyzed. Wash et al. conclude that participants' real-world behavior has only a weak correlation with their reported behavior.

The studies described here used samples of end users. However, experts, such as developers, differ from end users in their mental models, behavior, and so on [9, 105, 140, 141]. Therefore, this research will contribute methodological knowledge to security studies conducted with developers.

3.4. Security Developer Studies

Several studies of security have been conducted with samples of (professional) developers. For example, Gorski et al. [48] conducted a controlled online experiment with 53 GitHub users to test the effectiveness of API-integrated security advice. They found that 73% of participants who received a security warning and advice improved their insecure code. Acar et al. [49, 50] and Wijayarathna and Arachchilage [172]. also recruited GitHub users for security studies with developer samples. These and other studies of security using developer samples were introduced above in Sections 3.1.2 [49, 104, 139, 171, 172] and 3.2 [50, 51]. This section describes studies of security using developers that go beyond investigations of password storage or the usability of crypto APIs.

Prechelt [52] performed an exploratory study comparing Web development platforms (Java EE, Perl, and PHP). Nine teams comprising 27 professional developers had 30 hours to implement a Web-based application. The resulting applications were analyzed with regard to several aspects, such as usability, functionality, reliability, security, structure, etc. The findings suggested that wide differences in platform characteristics were presented in some dimensions, although possibly not in others. In comparison to Perl and Java EE, PHP was found to have the smallest within-platform variations. In [178], Stärk et al. conducted a further study comparing Web development platforms by asking teams of professional developers working with different programming languages (PHP, Java, Pearl and Ruby) to implement the same specification of a web application. The authors were able to confirm some of the conclusions from [52]. In [179], Prechelt and Stärk acknowledged that programming languages are rather a weak attribute to define a web development platform, because frameworks using the same language can result in different platforms as well as frameworks using different languages can result in the same platform.

In an effort to discover why programmers make security errors, Xie et al. [180] conducted 15 semi-structured interviews with professional software developers with experienced in Java, C++, C, Python, PHP, and JavaScript. This study observed a gap between participants' knowledge and awareness of software security and their reported security behavior. Most participants had adequate knowledge about security but could not list any precise measures of secure coding. The study concludes that developers need more interactive tools to help them implement security measures.

Redmiles et al. [181, 182] investigated security behavior in general and found that participants with IT skills accept security advice from different sources and reject advice for different reasons, such as handing responsibility to other actors.

Johnson et al. [183] conducted 20 interviews with professional developers and found that false positives and poor warning visualization are reasons for not using static analysis tools.

Thomas et al. [184] investigated how well developers interact with and how they perceive a tool suggested by the researchers. They recruited 13 participants with professional development expertise and conducted interviews as well as an observation-based study. The authors concluded that their tool is effective in communicating security vulnerabilities.

As outlined in Section 3.1.2, Balebako et al. [171] 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.

Assal et al. [53] conducted an online survey with 123 software developers to examine human factors which influence software security. In contrast to Balebako et al. [171], Assal et al. [53] 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. Overall, the authors found their participants to be self-motivated towards security and stressed the need to investigate organizational issues. The study presented in this thesis also examined whether the size or security focus of a company affected participants' decision to store user passwords securely, but no significant effect was found.

3.4.1. Students

While observations of studies conducted with students are assumed to apply to professional developers in the domain of software engineering [107, 108, 110, 111], it is not known for certain yet on whether this holds true for security developer studies [43, 49, 54, 112]. Still, 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 students. For example, Thomas et al. [185] 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.

Acar et al. [112] conducted a between-subjects laboratory study to examine the impact of different documentation resources on the security of code. Fifty-four developers were given a skeleton Android app, which they had to extend in four security-related tasks: the storage of data, the use of Hypertext Transfer Protocol Secure (HTTPS), the use of Inter Component Communication (ICC), and the use of permissions. Almost 90% of participants were students. For assistance, they had access either to (1) Stack Overflow, (2) books, (3) the official Android documentation, or (4) could freely choose which source to use. Their main finding was that programmers assigned to Stack Overflow produced less secure code. A results, which was supported by the work of Fischer et al. [186]. Fischer et al. analyzed 1.3 million Android apps and found that 15.4% of them contained Stack Overflow source code. Of the analyzed source code, 97.9% contained at least one insecure code part.

Jain et al. [187] 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. [188] 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. They, therefore, emphasized the need for improved error messages. Both Jain et al. [187] and Barik et al. [188] argued to specifically have recruited students to avoid the unrealized bias of professionals being familiar with APIs.

Layman et al. [189] 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. Moreover, Scandariato et al. [190] investigated the effectiveness of static analysis compared to penetration testing when performing security analysis of applications. They 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. [189] and Scandariato et al. [190] advised to recruit professional developers for future work.

3.4.2. Freelancers

In addition to students, it can be convenient to recruit freelancers for studies conducted with developers. For example, Yamashita and Moonen [191, 192] 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.

A further study with freelancers [139] was already discussed in Section 3.1.2. For the sake of completeness, that study will be presented here more detailed with regard to the methodology and the specific sample of freelancers. In a tech report, Bau et al. [139] 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 gap was found between freelancers' knowledge and the resulting implementation. For the presented security developer study conducted with freelancers in this thesis, the photo sharing scenario was used as the inspiration for the company scenario. However, the scope of the task was substantially reduced to focus solely on the registration process.

3.4.3. Student and Professional Comparison

In the domain of software engineering, there are multiple studies examining the applicability of students as proxies for developers, for example when studying requirement prioritization [108, 110], judging the lead-time of software development projects [107] or regarding the code quality of test-driven development experiments [111]. While there are caveats, on the whole it is considered acceptable to do exploratory research with students. Recent work indicated that this can also holds true in the field of security software engineering [43, 49, 54, 112, 117]. As outlined in Section 3.4.1, Acar et al. [112] conducted a lab study with 54 Android developers to explore the impact of information sources on code security. The participants included students as well as professional developers. The authors asked them to implement security and privacy relevant code while only having access to different information sources, like the official Android documentation. The authors found that professionals produced more functional code than students. However, with regard to security, no significant effect was found.

As it is often difficult to recruit professional developers for studies, Acar et al. [49] wanted to find out whether active GitHub users could be of interest for usable security studies (see Section 3.1.2). They conducted an online experiment with 307 GitHub users, who had to implement security-related tasks. One of these tasks considered credential storage. 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. [43] examined reasons for weak Transport Layer Security (TLS) configurations. They recruited students with expert knowledge and asked them to work on tasks regarding the deployment process for HTTPS. Later, they conducted interviews with professional security auditors to compare their statements to what the students had implemented. 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. [193] 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. [54] 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 [54] findings, the study results presented in this thesis showed that professionals' and students' security behavior was different in absolute terms. However, similarities were found with respect to relative terms.

3.5. Summary

The previous sections have summarized existing research on password-storage security, API usability, the ecological validity of studies, and studies of security using samples of professional, student, and freelance developers. While this literature review is not meant to be complete, it provides some context for the current studies that I have conducted and described in this thesis. They contribute to knowledge in this area as follows:

- Previous security studies have pointed out that many software developers fail to implement secure password storage, threatening a massive amount of sensitive user data. In order to gain a deeper insight into developers' perceptions and understanding of secure password storage, a qualitative study was conducted with CS students and is presented in Chapter 4.
- In contrast to previous research, the study conducted with CS students finds that the use of copy/paste can lead to secure code (Chapter 4).

- In order to extend knowledge of developers' password-storage related security behavior, quantitative studies were conducted with CS students, freelance developers, and professional developers working at different companies. These studies are presented in Chapters 5, 6, and 7.
- Research on API usability is extended through an examination of two Java APIs with varying levels of secure password-storage support. The findings are presented in Chapters 4, 5, and 7.
- Ecological validity is addressed in methodological considerations of different study tasks (security prompting vs. non-prompting) in Chapters 4 to 7; in a comparison of qualitative and quantitative findings in Chapter 5; and in a comparison of different developer samples (students, freelancers and professional developers employed by companies) used in a password-storage study (Chapter 7).

4. A Password-Storage Qualitative Usability Study with CS Students

Disclaimer: *The contents of this chapter were published as part of the publication "Why Do Developers Get Password Storage Wrong? A Qualitative Usability Study" (ACM SIGSAC Conference on Computer and Communications Security (CCS) 2017) [113]. This paper was produced in cooperation with my co-authors Anastasia Danilova, Christian Tiefenau, Marco Herzog, Sergej Dechand, and Matthew Smith. I designed and implemented the web application frame for the programming task, including the front end and the back end. During the design process for this study, I received advice from Anastasia Danilova, Sergej Dechand, and Matthew Smith. I set up and performed the study in the laboratory. Anastasia Danilova also assisted me with participant recruitment and supervision, and Christian Tiefenau provided technical assistance during the execution of the study. I prepared the interview design and conducted all the interviews alone. Anastasia Danilova, Christian Tiefenau, and I analyzed the study data. I discussed key insights of the work with Anastasia Danilova and Matthew Smith. Marco Herzog provided support on technical issues as the paper was being written. Finally, I prepared the paper for publication in cooperation with Anastasia Danilova, Christian Tiefenau, and Matthew Smith.*

4.1. Motivation

In the last decade usable security research has focused mainly on issues faced by end users. In their paper "Developers are not the enemy," Green and Smith [2] argue that usable security research needs to be extended to study and help developers who have to interact with complex security APIs. Storing and authenticating user login data is one of the most common tasks for software developers [104]. At the same time, this task is prone to security issues [2]. Frequent compromises of password databases highlight that developers often do not store passwords securely - quite often storing them in plain text [94, 95, 102, 103].

To gain an understanding on where and why developers struggle to store passwords securely, we conducted the first usability study of developer behavior when tasked with writing code to store passwords and authenticate users. Since this is the first work in this domain, we chose to conduct a qualitative study with the ability to conduct in-depth interviews to get feedback from developers. We were interested in exploring two particular aspects: Firstly, do developers get things wrong because they do not think about security and thus do not include security features (but could if they wanted to)? Or do they write insecure code because the complexity of the task is too great for them? Secondly, a common suggestion to increase security is to offer secure defaults. This is echoed by Green and Smith [2] who call for secure defaults for crypto-APIs. Based on this suggestion, we wanted to explore how developers use and perceive frameworks that attempt to take the burden off developers. We did not find any framework, which offers safe defaults for password storage. The highest level of support we found is *opt-in* [119].

To offer insights into these questions, we conducted three pilot studies and a final 8-hour qualitative development study with 20 computer science students. The recruitment of computer science students for software engineering studies is considered an acceptable proxy for developers [107, 108, 110, 111]. The students were given a role-playing scenario in which they were tasked to include registration and authentication code into an existing stub of a social networking application. To gather insights into task framing, half of the students were told to store the passwords securely and the other half were not. This latter group of students would have to decide on their own volition whether, since they were writing code for storing user passwords, they should store them securely. Also, one half of each group was given the Spring Framework to work with, which contains helper classes for password storage, while the other half had to use the Java Server Faces (JSF) Framework in which they would have to implement everything themselves.

4.2. Study Design

A detailed description of the study design is presented in Section 2.1. Most importantly, we designed a study through which we could gain insights into the development process of end-user password storage as well the effect of telling participants that we want them to store the end-user passwords securely (prompted group) as opposed to them having to think of security on their own (non-prompted group).[1] Participants were advised to use two different Web frameworks with different levels of support for secure password storage.

In order to get substantive results we used multiple data-collection approaches for the study: (1) an implementation task which was logged in its entirety, (2) a survey, and (3) a semi-structured interview. Since we were interested in qualitative data, we opted to conduct the study in a laboratory setting instead online. Although online studies have the advantage of letting people work in their natural environment, it is harder to monitor all their activity and the in-person interview is harder to conduct. First, in our usability lab, we provided the setting, including software and hardware, and thus were able to track all actions of the participants. Second, we had full control of time constraints and could avoid any kind of unnecessary disturbance during the implementation task. Finally, we could ensure that participants did not get outside help without such assistance being logged and the

[1]In [113], the term "priming" was erroneously used to describe security prompting. A detailed description of the difference between security priming and security prompting is provided in Section 2.1.3.

							How familiar are you with Not familiar at all (1) - Very familiar (7)				
	Gender	University	Study program	Age	Nationality	Semester	Java	PostgreSQL	Hibernate	Eclipse IDE	Total skills
JN1	Male	University of Bonn	MSc Computer Science	NA	Bangladeshi	8	6	4	4	6	20
JN2	Female	University of Bonn	MSc Computer Science	23	Pakistani	3	3	1	1	6	11
JN3	Male	University of Bonn	MSc Computer Science	25	Uzbek	2	5	3	2	5	15
JN4	Male	University of Bonn	BSc Computer Science	23	German	6	5	2	1	6	14
JN5	Female	University of Bonn	MSc Computer Science	27	Indian	5	5	4	1	6	16
JP1	Male	University of Bonn	MSc Computer Science	25	Chinese	5	5	1	1	4	11
JP2	Male	University of Bonn	BSc Computer Science	22	German	4	6	6	1	6	19
JP3	Male	University of Bonn	MSc Computer Science	26	Iranian	4	4	2	2	6	14
JP4	Male	Aachen University	MSc Media Informatics	27	Indian	2	4	2	1	3	10
JP5	Male	Aachen University	MSc Media Informatics	25	NA	2	2	1	1	2	6
SN1	Male	University of Bonn	MSc Computer Science	24	German	10	6	4	1	5	16
SN2	Male	University of Bonn	BSc Computer Science	20	German	2	7	5	2	4	18
SN3	Male	University of Bonn	BSc Computer Science	24	German	8	6	3	1	6	16
SN4	Male	University of Bonn	MSc Computer Science	25	Syrian	3	7	5	7	7	26
SN5	Male	University of Bonn	BSc Computer Science	19	German	2	5	4	1	4	14
SP1	Male	University of Bonn	MSc Computer Science	25	NA	4	4	3	2	4	13
SP2	Male	University of Bonn	MSc Computer Science	25	Syrian	4	6	3	4	4	17
SP3	Male	University of Bonn	BSc Computer Science	20	German	2	5	3	1	4	13
SP4	Male	University of Bonn	BSc Computer Science	25	German	10	5	3	1	5	14
SP5	Female	University of Bonn	MSc Computer Science	NA	Indian	4	5	4	4	6	19

Table 4.1.: Participants' demographics
S=Spring, J=JSF, P=Priming, N=Non-Priming

record available for analysis.

Participants were briefed about the procedure and purpose of the study and told that they could quit at any time. Half the participants were told that the study was about the usability of Java frameworks, while the other half were told the study was about secure password storage. Thus, half the participants were subjected to a deception task. For those with the explicit task we did not need to use any deception. These participants were told the true purpose of the study in the **Introductory Text** and in the **Task Description** (see Appendix A.5, A.6, and A.7). Both groups completed a short pre-questionnaire about their expectation of how difficult the task would be and a self-assessment of their programming skills. After finishing the implementation task, participants completed a further survey and were interviewed about their implementation, their experience solving the task, and their thoughts on the usability of the framework they worked on. Finally, both groups were debriefed, with the non-prompted participants additionally being informed about the true purpose of the study.

4.2.1. Participants

Researchers conducting computer science studies with professional developers often encounter obstacles, such as high costs and low response rate [109]. In the domain of software engineering, there are multiple studies examining the applicability of students as proxies for developers [107, 108, 110, 111]. While there are caveats, on the whole it is considered acceptable to do exploratory research with students. Furthermore, the recent work of Acar et al. indicates that this holds true in the field of security software engineering [49, 112].

Therefore, we recruited 7 Bachelor and 13 Master students of computer science via the computer science mailing list and posters in the CS buildings. In the invitation text, we did not explicitly mention security (see Appendix A.1). Participants' demographics are available in Table 4.1. All applicants filled out a short questionnaire about their familiarity with Java and the APIs used in the study (see Appendix A.2). Thus, we were able to calculate their programming language and used APIs skill levels. These are also presented in Table 4.1. As a prerequisite for our study, familiarity with Java and the integrated development environment (IDE) Eclipse were required. Of the possible 28 points, no participant reported to be very familiar with all APIs and Java.

Since the Bachelor program in Computer Science at the University of Bonn is completely in German and the Master program is in English, we gave our participants the opportunity to decide which language they wished to use for the study, including the written study task and the interview.

4.2.2. Task Design

The detailed description of the programming task can be found in Section 2.1.2. We asked participants to imagine they were part of a team working on creating a social networking site for our university. While our aim was not to quantitatively compare variables, we nonetheless studied four variants on how the study was conducted (see Table 2.1).

Since there is still very little knowledge on how to design tasks for developer studies in the security realm, we conducted three pilot studies to refine the task design. The following three subsections briefly describe differences and improvements of the pilot studies.

Pilot Study 1

In order to test our task design, we conducted a pilot study with 4 participants: 2 Computer Science PhD students of the Usable Security and Privacy group at the University of Bonn, and 2 Computer Science Master students in their 1st and 3rd semester at the University of Bonn. The main purpose of the first pilot study was to test timing, the understanding of the task, and to see if the students managed to complete the task. In this pilot study we only tested the prompting version of the study since, a) the students knew our group and could not have been fooled into believing the non-prompting scenario, and b) the three measurement goals can more reliably be achieved if students definitively attempt the security part of the task.

We set the overall time threshold of the study to 8 hours. This time included a reception phase, an implementation task, a survey, an interview, a debriefing session, and a break of half an hour. Participants were compensated with 80 euros (10 euros per hour). We also asked our participants to “Think-Aloud”, since this is a common method in end-user research to gain insights into the thoughts of participants.

Participants had to write the code for the **registration** and an **authentication** part of the Web application with one of the two Web frameworks: Spring (PhD 1 and Master student 1) or JSF (PhD 2 and Master student 2). Registering an end user means storing the user’s data, which she/he provides through an interface to a database. Authenticating the end-user means to log in the user with her/his credentials provided in the registration process.

Due to hardware problems, we had to exclude one participant (PhD student 2) from the study analysis. Participants indicated high skill levels in Java programming and Web development. On average, PhD student 1 and the first Master student needed about 7 hours to solve the task. The second Master student did not implement a functional solution for the **authentication** task within the 8 hours.

Based on our observations, we decided that our first design interaction was too complex and that students had difficulty completing the task in time. Because of this we decided to remove the authentication task and to focus entirely on password storage during the registration process.

We also found that, unlike in end-user studies, the Think-Aloud method did not provide us with valuable results. Most of participants did not talk at all during programming. If they were talking, they only addressed topics apart from our security focus. They stated obvious things, such as searching for a term, or items which were discussed in the interview after solving the task anyway. It seems that both the length and complexity of the task makes Think Aloud problematic. We therefore removed this element of the study. This also opened up the possibility of having multiple participants conduct the study at the same time.

All further studies were executed in our usability lab with 9 participants and 1 supervisor. The workspaces were set up in a way that participants could not see one another's work and they were equipped with noise canceling headphones to minimize distractions.

Pilot Study 2

In the second pilot study, participants had to implement the registration part of the Web application. As before, we only used the prompting scenario. We invited 2 Computer Science Master students from the University of Bonn and 1 Computer Science Master student assistant from the Usable Security and Privacy group of the University of Bonn. While the student assistant was compensated with 60 euros (we expected that the implementation of the registration task would not exceed 6 hours), the Master students worked on the task in the course of a Usable Security and Privacy Lab. Thus, they were compensated with ECTS-credits[2].

On average, participants took 5 hours and 30 minutes to complete the study. We observed that the database connection in particular was costing the participants a lot of time. Since this was not our primary interest, we added some hints for this part of the task so participants would not get lost in it.

We estimated that the streamlined study would last about 4 hours. Since we wanted to recruit as many students as possible, we announced the next phase of the study as lasting approximately 4-5 hours, 8 hours max. (with a break of half an hour) and announced a payment of 60 euros. As the next subsection will show, this turned out to be a mistake.

[2]Participation was still voluntary and the student could have chosen a non-study task to comply with ethical guidelines.

Pilot Study 3

We invited 7 participants to what was supposed to be the first round of the main study. They were randomly assigned to the prompting/non-prompting and Spring/JSF scenarios. On average participants took part in the study for about 5 hours and 30 minutes. However, we noticed that, unlike in the 8-hour pilot study, participants who did not complete the task within the estimated 4-5 hours got frustrated and gave up. We received non-functional and insecure solutions, and the interviews indicated that timing was a major factor for this. Also, unlike in the previous two groups, some participants did not know how to verify that they had solved the task. An interesting effect occurred in the non-prompted group. After solving the task, participants were asked to complete a survey containing questions about secure password storage. Upon reading the questions, some participants went back into the code and implemented secure storage.

Due to these critical issues, we discarded the results from these 7 students and created another iteration of the study design. Firstly, we revised the task description to make it clear when the task was functionally complete: "*You have solved the task when you can store user data in the table appuser using the User Registration Form.*" Additionally, the prompted group received the instruction: "*Please ensure that the user password is stored securely.*" The final version of the task description and implementation hints can be found in the Appendix A.5, A.6, A.7. Secondly, we revised the time constraints on the notices for the study: *"The study will last 8 hours."* We decided to allow students to go back into the code after reading the questions in the survey but built in logging mechanisms to track when this happened. This actually gave us very valuable data that showed that the participants had the skills, but just did not think they were necessary.

To give stronger incentives, we increased the payment for participation to 100 euros. Additionally, we provided snacks and refreshments to ensure participants' contentment. Previously, students had been asked to bring their own food.

After this third pilot study the task design worked well, with the 20 students completing the study as planed. Our study was conducted in May 2017. The pilot studies were conducted in December 2016 and April 2017.

4.2.3. Survey and Interview

Once the implementation task had been finished or abandoned, participants had to fill out an exit survey. After that they were interviewed about their task solution and their knowledge about end-user password storage security mechanisms. The interview and survey questions can be found in the Appendix A.3 and A.4.

Survey

For the survey we applied the suggested approach of [194] to use seven-point rating scales instead of five-point scales. We asked participants for their demographics and programming experience. They also had to indicate their experience with the APIs employed in the study.

In order to address user satisfaction, we wanted to find out how easy or difficult the implementation task was perceived by participants (experience) in comparison to how easy or difficult they thought it was going to be (expectation), a method suggested in [195, 196]. Therefore, we asked participants a *Single Ease Question* (SEQ) [197, 198] after they had read the study task description, but **before** they had started to work on the implementation task. Additionally, they had to answer the SEQ **after** they had finished the implementation task.

Interview

After participants had completed the exit survey, they were interviewed about their implemented solutions and end-user password storage security mechanisms. They were also asked what they would do if they were working for a company and were given exactly the same task with the same description and time constraints. Specifically, they were asked whether they would have solved the task in the exact same way as they solved it in the study.

Regarding the prompting and non-prompting scenarios, both sets of interview questions sets were similar apart from those questions illustrated in Table 4.2. The prompted participants, who were told to solve their task securely, were asked whether they would have done this if they had not been explicitly requested to do so. The non-prompted participants were asked about security awareness in any case, regardless of how they solved the task.

A single trained researcher conducted all 20 qualitative interviews. The interview sessions were audio-recorded. After the recordings were transcribed, two researchers independently coded all interviews using Ground-Theory and compared their final code books using the inter-coder agreement. The Cohen's [199] kappa coefficient (κ) for all themes was 0.81. A value above 0.75 is considered a good level of coding agreement [200].

4.2.4. Evaluating Participants' Solutions

For the usability analysis of the Spring and JSF APIs, we used the ISO usability specification as a basis, i.e., *"the extent to which a product can be used by specified users to achieve specified goals with **effectiveness, efficiency and satisfaction** in a specified context of use"* [201]. Both the *survey and the interview* covered the satisfaction aspect. To specify the efficiency, we measured the *time* a participant needed to solve the task. For specifying the effectiveness, we examined the *functionality and security* of the participant's program code. A detailed description of participants' code evaluation is available in Section 2.2.

	Participant stated to have stored the end-user password... *...securely*	*...not securely*
Prompted Group	"Do you think you would have stored the end-user password securely, *if you had not been told about it*?"	-
Non-Prompted Group	"*How did you become aware* of the necessity of security in the task? At which point did you decide to store end-user password securely?"	"*Were you aware* that the task needed a secure solution?"

Table 4.2.: Questions asked depending on the prompting/non-prompting scenarios.

If the end users' data was stored to a database, then the task was solved functionally to an adequate degree. To fulfill password-storage security, participants had to hash and salt the end-user passwords [161]. Accordingly, a *security scale*, which was introduced in Section 2.2.1, was used for analysis. The score from 0 to 7 shows how well the participant' solution is with regard to state-of-the-art password storage practices. It must be stressed, though, that no participant received all 7 points and thus did not become aware of recent academic results showing that memory-hard hashing functions are necessary for best security [153, 157].

4.3. Ethics

Our institution does not have a formal institutional review board (IRB) process for computer science studies, but the study protocols were cleared with the projects ethics officer. Our study also complied with the strict German privacy regulations. The data was collected anonymously and our participants were provided with a consent form that had all information on their recorded data. The participants were informed about withdrawing their data during or after the study.

4.4. Results

We report statements of specific participants by labeling them JP/JN/SP/SN, from 1 to 5 each, according to their scenarios from Table 2.1 (see Section 2.1). Table 4.3 summarizes participants' security expertise identified from the exit survey. The maximum security expertise to achieve is 28. Participants' security expertise ranged from 8 to 23.

15 participants stored passwords before working on the study task. Despite this fact, many of them answered, in the question concerning their knowledge of secure password storage, that they have

Participant	**Security Expertise**[1]	**Knowledge of Secure Password Storage**	**Stored Passwords Before**	**Framework supported me**	**Framework prevented me**
	Not knowledgeable at all (4) - Very knowledgeable (28)	Not knowledgeable at all (1) - Very knowledgeable (7)		Strongly disagree (1) - Strongly agree (7)	Strongly disagree (1) - Strongly agree (7)
JN1	15	3	✗	4	4
JN2	12	2	✓	4	4
JN3	16	5	✗	3	3
JN4	21	6	✓	4	4
JN5	13	3	✓	4	4
JP1	13	2	✓	3	1
JP2	23	5	✓	2	4
JP3	19	6	✓	3	2
JP4	15	2	✓	1	5
JP5	13	2	✓	2	6
SN1	16	6	✓	1	1
SN2	23	5	✓	1	4
SN3	17	4	✓	4	4
SN4	19	7	✗	1	7
SN5	10	1	✗	4	4
SP1	15	4	✗	7	1
SP2	14	5	✓	1	3
SP3	8	2	✓	7	1
SP4	12	4	✓	7	2
SP5	16	4	✓	4	4

Table 4.3.: Participants' security expertise, password knowledge, and framework support
S=Spring, J=JSF, P=Prompting, N=Non-Prompting
[1]We determined the security expertise from Q1-Q4 (see Appendix A.3.1). We combined the answers of the participants to Q1-Q4 by adding up the numbers of their responses, whereby the answer for Q2 had to be reversed for consistency reasons.

little understanding of this topic. Further, participants identified their experience with the framework, which is discussed in Section 4.4.5 in detail.

4.4.1. Functionality and Security

Table 4.4 summarizes all solutions from our participants in terms of functionality, security, and implementation time. Not a single participant from the non-prompted groups, either working with JSF or Spring, stored the end-user passwords securely in the database. The participants' implementation time in the non-prompted groups varied from about 2 to 6.5 hours. From the prompted groups, SP1, SP3, SP4, and JP4 were able to store the password securely with a security score of 6. However, none of the participants used a memory-hard hashing function and achieved all 7 points. SP1, SP3, and SP4 used `bcrypt` with a 192-bit password hash and a 128-bit salt. Their implementation time varied from about 3 hours to 7 hours. All of them made use of Spring Security's `PasswordEncoder` interface.

JP2 and JP4 both used `PBKDF2` with different algorithms, iteration counts and salt lengths. Their implementation time varied from about 3 hours to 4 hours. JP2 employed `SHA256` with a hash length of 512 bits, 1 000 iterations and a 256-bit salt. JP4 used `SHA1` with a hash length of 160 bits, 20 000 iterations and a smaller salt of 64 bits. By contrast, JP3 used the hash function `SHA256` with only one iteration and without salting.

Some participants changed the length constraints of the password. JP1 and JP5 used 3-50 characters as length constraint for the end-user password. SN4 and SP1 decided on a password length greater

	Time	Functionality	Security					
			Hashing			Salt		
	(hh:mm)	Storage working	Hashfunction (at most +2)	Length (bits) (+1 if ≥ 160)	Iteration count (at most +1)	Used library (at most +2)	Length (bits) (+1 if ≥ 32)	**Total** (7)
JN1	06:26	✗						0
JN2	04:05	✓						0
JN3	03:01	✓						0
JN4	04:11	✗						0
JN5	05:30	✗						0
JP1	04:55	✓						0
JP2	03:12	✓	PBKDF2(SHA256)	512	1 000	SecureRandom	256	5.5
JP3	05:29	✓	SHA256	256	1			2
JP4	04:12	✓	PBKDF2(SHA1)	160	20 000	SecureRandom	64	6
JP5	06:32	✓						0
SN1	03:15	✓						0
SN2	02:24	✓						0
SN3	02:01	✓						0
SN4	04:01	✓						0
SN5	04:50	✓						0
SP1	03:15	✓	BCrypt	184[1]	1 024	Spring Library	128	6
SP2	01:54	✓	MD5	128	1			1
SP3	07:00	✗	BCrypt	184	1 024	Spring Library	128	6
SP4	03:39	✓	BCrypt	184	1 024	Spring Library	128	6
SP5	03:44	✗						0

Table 4.4.: Evaluation of the implemented solutions
S=Spring, J=JSF, P=Prompting, N=Non-Prompting
[1]In [113] 192 bits were reported as the digest size for `bcrypt`. Here, the digest size of `bcrypt` was changed from 192 bits to 184 bits, reasonable by practical implementation standards.

than 6 characters, while SN1, SN2, and SP5 chose a password length constraint with a minimum of 8 characters.

4.4.2. Choice of Hash Function

Our participants chose hash functions based on different factors. SP1 noted that he chose to work with `bcrypt`:

> "A lot of forums say that is very secure and very up to date and has a high vote in some cryptography website. [...] I find it trust-able, the Stack Overflow and cryptography forums."

In contrast, JP2 decided to not use `bcrypt` as he did not trust the library source. Further reasons for choosing a library were past experience. SP2 decided on `MD5` because he had used it before. JP4 would have also used `MD5` if the task had not asked for security:

> "I would have used `MD5` because I was already comfortable with it and so I wouldn't have thought it had vulnerability."

4.4.3. Expectations and Experiences With the Task

In order to evaluate the task difficulty before and after implementation, we asked our participants to rate their expectations and experiences. Table 4.5 summarizes the before and after ratings. We

	Expectation	**Experience**
	Very difficult (1) - Very easy (7)	
JN1	5	5
JN2	4	4
JN3	3	4
JN4	5	3
JN5	3	2
JP1	3	2
JP2	4	5
JP3	5	5
JP4	2	4
JP5	4	2
SN1	4	6
SN2	5	4
SN3	3	5
SN4	5	5
SN5	3	3
SP1	4	4
SP2	6	5
SP3	3	3
SP4	4	5
SP5	2	3

Table 4.5.: Participants' expectation and experience of the task difficulty
S=Spring, J=JSF, P=Prompting, N=Non-Prompting

additionally report the mean values of expectation and experience with regard to the prompting/non-prompting and framework support scenarios. However, these values have to be regarded with care since the main purpose of qualitative research is to explore a phenomenon in depth and not to generate quantitative results[3]. On a seven-point rating scale (1: Very difficult - 7: Very easy) the mean values for the non-prompted group were 4 before and 4.1 after working on the task. The mean values for the prompted group were 3.7 before and 3.8 after working on the task.

Before the implementation phase, our prompted participants rated the task difficulty slightly more difficult than the non-prompted participants. This might indicate that our participants acknowledged the security overhead to increase the difficulty of the task. This bears further examination in the future since such a perception might make developers shy away from security code.

Comparing the mean values of the framework support scenarios, the groups had nearly similar mean values (Spring: 3.8, JSF: 3.9) before the task implementation. After the task, however, the mean value of experience changed to 4.3 for Spring and 3.6 for JSF. This might indicate that the participants perceived solving the task with JSF as more complex than solving it with Spring.

Further, we observed that participants who implemented solutions with a low security score (JP3, SP2), e.g., because of using weak hash functions, rated the difficulty of the task as rather low. SP2 stated that he used `MD5` because he had already used it in previous applications. His self-reported

[3]It should be considered that our sample is small and not all groups indicate a normal distribution. The individual values can be found in Table 4.5.

	Secure storage		Hashing		Salting		Company	
	Security Score (7)	Own Opinion	Basic Understanding	Deeper Understanding	Deeper Understanding	Believed Have Used	Similar Solution	Further Code Improvement
JN1	0	✗	✗	✗	✗	✗	✓	✓
JN2	0	✗	✓	✗	✗	✗	✗	✓
JN3	0	✗	✓	✗	✓	✗	✗	-
JN4	0	✗	✓	✓	✓	✗	✗	✓
JN5	0	✗	✓	✗	✗	✗	✗	✓
JP1	0	✗	✓	✗	✗	✗	-	-
JP2	5.5	✓	✓	✓	✓	✓	✗	✓
JP3	2	✓	✓	✓	✓	✓	✓	✓
JP4	6	✓	✓	✓	✗	✓	✓	-
JP5	0	✗	✓	✓	✗	✗	✓	-
SN1	0	✗	✓	✓	✓	✗	✓	-
SN2	0	✗	✓	✓	✓	✗	✓	✓
SN3	0	✗	✓	✓	✓	✗	✓	✓
SN4	0	✗	✓	✓	✓	✓	✗	✓
SN5	0	✗	✓	✓	✗	✗	✓	✓
SP1	6	✓	✓	✓	✓	✓	✓	-
SP2	1	✓	✓	✓	✓	✗	✗	✓
SP3	6	✓	✗	✗	✗	✓	✓	✓
SP4	6	✓	✓	✓	✓	✓	✓	✓
SP5	0	✗	✓	✗	✗	✗	✓	✓

Table 4.6.: Participants' security knowledge and code self-assessment
S=Spring, J=JSF, P=Prompting, N=Non-Prompting

familiarity with the task could explain his rating. All other participants who were able to securely implement the task indicated it as more difficult.

We observed no noticeable trends in the expected difficulty and performance of the participants. The participants from the prompted group who added (limited) security gave a similar rating (JP3, SP1, SP3) or even a slightly better rating (JP2, JP4, SP2, SP4) after implementation. However, the participants who were not able to store passwords securely rated the task slightly more difficult (JP1, JP5) after the implementation phase. This indicates that they underestimated the complexity of the security part of the task.

4.4.4. Code Self-Assessment vs. Actual Security

The participants' code self-assessment sometimes differed from the solutions they actually implemented. Table 4.6 summarizes the security knowledge of each participant and the participants' self-assessment regarding their own solution we learned from the interviews compared to our security score. Furthermore, their estimations on whether they would solve the task differently in a real company are listed in the table.

Lacking knowledge of hashing and salting does not necessarily lead to insecure code, nor vice versa. With the exception of SP3 and JN1, all participants presented at least a basic knowledge of hashing. A number of participants (JN1, JN2, JN5, JP1, JP4, JP5, SN5, SP3, SP5) did not know the definition of salting. The other participants explained that salting prevents usage of rainbow tables, thereby adding an additional level of security. While JP3 and SP2 assumed that they implemented the password storage securely, they in fact did not. SP3 was not able to explain hashing or salting correctly but managed to store the passwords securely by using the Spring helper classes. He stated:

> "Every time you put a password in it, e.g. 'password', you get each time another hash."

Although his implementation was not functional and his understanding of hashing not correct, he was able solve the task securely. By contrast, the participants from the non-prompted Spring group all showed a deeper understanding of hashing during the interviews, but did not store the end-user password securely. Also, in the prompted Spring group, SP2 for instance could explain hashing and salting properly, but only gained a security score of 1/7.

What is more, in our pilot studies some participants mentioned that they would have implemented a different solution if the task was assigned in an actual company. This is a problem many studies face. To get insights into this we asked our participants whether their solution from the study would differ from the one they would implement for a company.

Almost all participants stated that they would have solved the task in a better way. SP3 and JN5 stated that in a company they would ask for help. Interestingly, some participants explained that there would be a security expert or supervisor who would check for and require security:

> "I would ask my supervisor about it. [...] There is definitely another person that understood these kinds of things" (JN3).

The participants expected somebody to set security as a requirement or explicitly request it. SN3 and JN4 mentioned that the security of their solutions depend on the company:

> "It depends on the company. If it had been a security company I would have thought of something because they would have minded" (SN3).

JN4 goes even further saying that it depends on the data the company is handling:

> "Depends on the company data. If the data would be e.g. medical data, there would be other requirements" (JN4).

SP2 also thinks the responsibility lies with the employer or the company itself. He stated:

> "If the employer did not really constrain the programmer with certain requirements, then the programmer will do the work in, let's say, not in a sufficient way."

Other participants stated that if they had more time to get used to working with the framework, they would be able to produce a better solution.

4.4.5. Experienced Spring/JSF Support

Table 4.3 shows the agreement of all participants on whether the framework they used supported or prevented them from storing passwords securely. Seven of the non-prompted participants specified an undecided rating in the middle of the scale. SN3, for example, augmented his answer with these words:

> "I would say, that I'm undecided or don't know if it would have supported me, because I didn't look for it"

And to the prevention-related question he stated: "Actually the same answer."

Particularly interesting are the answers of participants belonging to the prompted group regarding the framework support. While 3 of the 5 participants who were using Spring, managed to store the passwords securely, and thus, agreed that Spring supported them (SP1, SP3, SP4), all participants from the prompted group using JSF disagreed that JSF supported them. Most participants that used JSF even answered that the framework prevented them from storing the password securely. JP4 mentioned:

> "I thought of finding something within the JSF Framework or within a Java framework which easily gives a hash function or they can use it easily but I couldn't get anything."

SP1, for example, in answering why the Spring Framework supported him, said:

> "Because it has the built in library `bcrypt` for example and some strong validation framework for the password."

This observation could suggest that developers, who are told that a secure storage is a requirement, first look for a library in the framework they are working with and - if they find it - are willing to use it for the task.

Other participants gave Spring as the reason for why they did not manage to store the end-user password securely. SN4 explained:

> "Because there are too many possibilities to implement user password hashing and Spring works in a way that it uses an authentication manager or something specific to Spring and one needs to extend the application to create this encryption or hashing schema."

SN4 even had a bad experience with Spring in his past job and stated that it was the main reason he quit the job. Further, SN2 felt that Spring hindered him from storing passwords securely because he did not know where to include his code into the framework. Another participant noted that JSF supported secure password storage "Because it provides that validation thing" (JN5).

Some participants were not familiar with Hibernate and experienced difficulties with inserting data from the form to the database (JP2, JN5). JP3 had problems with MVC in general, and had to work into it before working with it. Further, some participants needed more time to get familiar with the frameworks (SN4, SN1).

4.4.6. Prompting vs. Non-prompting Scenario

None of the 10 non-prompted participants implemented a secure solution. The reasons for this outcome are many and varied. Most of the participants mentioned that the task description did not require it, or rather they did not know it was a requirement (JN2, JN3, JN5, SN1, SN2, SN3, SN4, SN5). While this may seem obvious, there is still a potentially important lesson to learn from this. Even though we would expect that developers should realize that storing passwords is a security critical task, some did not even make that connection, as the following quotes indicate:

> "Umm, actually literally when I was in the project I didn't feel much like that it was related to security." (JN5)

> "I thought the purpose of the study is not about security so I didn't do it." (JN2)

Even if JN5 was aware of security in general he did not attempt to store the end-user password securely, because his task did not explicitly ask him to do so. Others did not think of security at all. For instance, JN2 noted:

> "It was like just to store information in tables and I didn't notice that, I should store password securely."

During the completion of the exit survey, some participants recognized that secure password storage might be part of their task. Participants were asked, whether they believe they had stored the end-user password securely. JN2, e.g., noted:

> "When I completed the task until then I didn't notice this, but when I started filling the form, the survey form at the end of the task, then I felt that I missed this point."

What is more, one non-prompted participant struggled with the Spring Framework. He reasoned it was hard to find a way to solve the task securely "because there are too many possibilities to implement user password hashing..." and that he "would need more time to get familiar with the system" (SN4).

By contrast, 7 of 10 prompted participants at least hashed the passwords and stored them on the database. SP1 explained that even if he would not have been prompted he would have stored the passwords securely, since his colleague experienced the downsides of insecure password storage. SP2 also claimed that he would have stored the end-user passwords securely even if he had not been told to do so. However, he only used `MD5` to secure the passwords. Finally, when JP1 was asked why he stored the password in plain text, he gave this candid answer:

> "I'm lazy."

This represents one of the main findings of our work. Even in a task so obviously security-related, participants who were not explicitly told to securely store the end-user passwords did not. While part of this effect can be explained by the fact that the participants knew that they were in a study, it is still important to note that not all of our participants linked password storage to a security-sensitive task. Thus we find that, similar to misperceptions end-users have about their password behavior, developers have misperceptions of their own, which we need to combat. Section 4.4.7 goes into more detail on this matter.

Another important take-away is that developer studies need to take care when framing their tasks, since we saw a very large effect. Giving developers small focused tasks concerning crypto libraries will not reveal all the problems we saw in the more realistic context of complex tasks which mix security and functionality development. Both types of studies are worthwhile; however, the decision needs to be taken consciously according to the goal of the studies.

4.4.7. Misconceptions of Secure Password Storage

When we asked our participants what storing passwords securely means, some stated that the passwords need to have certain constraints and be validated for security, e.g., character length (JN5). It became clear that some participants assume that securing passwords includes validation of the passwords on the end-user side instead of secure storage in the database on the developer side. What is particularly interesting is that developers were applying end-user security advice but not developer security advice. Another interesting finding is that one participant said that her solution is optimal because she tried to switch on SSL and secure the password transmission. JN1 saw no reason for hashing the passwords, stating:

> "I assumed that the connection will be a secure connection like with an HTTPS connection, so everything should come encrypted."

Further, some participants had never heard of, let alone could explain, hashing or salting a password (JN1, SP3).

SP2 was sure that `MD5` is secure. He was not aware of the consequences of using the broken hash function and even stated:

> "`MD5` with complicated salt is pretty secure."

4.5. Discussion and Key Findings

Developers are responsible for the software they implement. Interestingly, a number of our participants did not feel responsible for security when it was not explicitly mentioned. This is a surprising similarity to end users who see security as a secondary task. Getting developers to care about security is essential in order to produce software which protects its users. However, our study shows that there is still much we do not know on why developers behave the way they do. In particular, the insights gained from the qualitative approach proved to be enlightening to us since it goes beyond simply quantifying where the problems lie.

Some participants used weak hash functions like `MD5` or `SHA256` without key stretching simply because they were either not aware of the issues or had no negative experiences with them.

People tend to use the hash functions, they are familiar with - even if they have heard about more secure hash functions. Although in general non-prompted participants did not consider secure password storage, most of them showed a deeper understanding of it when prompted in the interviews. They were able to explain in detail how they would hash and salt an end-user password. Additionally, most participants were able to identify hash algorithms that were less recommended by referring to community discussions.

None of our participants used a memory-hard hashing function to store the end-user passwords securely, even though Spring Security offers an implementation of `scrypt` and participants could have used it just as easily as `bcrypt`. This highlights the need for academics to intensify the push of research results into tutorials, standardization bodies, and frameworks. Another solution worth exploring is removing less secure options from libraries. This solution needs to be studied with care since it creates issues of backward compatibility which might create worse security problems in the long run. However, the fact that not a single participant used state of the art memory-hard hashing functions is a troubling discovery and one that calls for urgent action.

Secure password storage was regarded from different perspectives in our study. Some participants thought their job was to ensure that end users behaved securely, e.g., by setting constraints to password length and strength. Other participants thought that using encrypted transmission would achieve secure password storage. This highlights that developers have similar security misconceptions as end users. Even though developers show greater theoretical knowledge of security concepts, this is no guarantee that they will use them or even acknowledge their necessity in their software.

Overall, our study indicates that, even when frameworks with better security support can help, prompting is essential to produce secure applications. Developers need to be told explicitly about what kind of security to add and they need help in avoiding outdated security mechanisms.

Overall, our key insights are as follows:

- **Security knowledge does not guarantee secure software:** Just because participants had a good understanding of security practices did not mean that they applied this knowledge to the task. Conversely, we had participants with very little security knowledge who created secure implementations.

- **More usable APIs are not enough**: Even when APIs offer better support for developers with ready-to-use methods which implement secure storage correctly, this is not enough if these methods are opt-in, as they were with all password storage APIs we found.

- **Explicitly requesting security is necessary**: Despite the fact that storage of passwords should self-evidently be a security sensitive task, participants who were not explicitly told to employ secure storage stored the passwords in plaintext.
- **Functionality first, security second:** Very similarly to end users, most of our participants focused on the functionality and only added security as an afterthought - even those who were prompted for security.
- **Continuous learning:** Even participants who attempted to store passwords securely often did so insecurely because the methods they learnt are now outdated. Despite knowing that security is a fast moving field, participants did not update their knowledge.
- **Conflicting advice**: Different standards and security recommendations make it difficult for developers to decide what is the right course of action, which creates frustration.

Since this is also one of the first studies of security APIs, we also gained several insights into running such studies which might be helpful for the community.

- **Think Aloud:** While Think Aloud is considered a good method to gather insights into study participants' mental process in end-user studies, we did not find it as useful for developer studies. Both the length and the complexity of the tasks seem to be a hindrance. In addition, evaluating Think-Aloud data is very work intensive.
- **Task framing:** Developer studies need to take great care when framing their tasks. Giving developers very small focused tasks concerning crypto libraries will not uncover most of the problems we saw in the more realistic context of complex tasks, which mix security and functionality development. Both types of studies are worthwhile; however, the decision needs to be taken consciously according to the goal of the studies.
- **Qualitative research reveals essential insights:** Prior research often focused on only examining the end-solution and analyzing survey responses in developer studies. We found that qualitative investigations can dig deeper and reveal misconceptions and lack of knowledge even if the solutions are secure.

4.6. Limitations

Below, we discuss the limitations of our study.

Population: Firstly, we recruited only computer science students instead of real developers. While related work from the field of empirical software engineering shows that students can be good proxies for real developers, it is still important to remember that there are differences between the two groups. Particularly, we want to underline that a lot of software development skills come from experience, which can be markedly absent from students. Secondly, all students were recruited from the same CS environment. While such a homogenous group enables a better comparison between the different conditions, it limits generalizability. Thus, no generalizability is claimed. Thirdly, despite 20 participants being a good number for a qualitative study, the high variability of skill and experience levels means that our results need to be backed up by follow-up work with larger scale studies.

Laboratory environment: The laboratory environment which allowed us to gather very rich and in-depth data also introduced an environmental bias. Participants knew they were part of a study and not working with real data. We used the common technique of role-playing to counter this as best we could; however, some participants nonetheless self-reported that they would have behaved more securely if they were given the same task in a company. We cannot quantify the effective size of this bias, since it is also possible that such claims were made out of embarrassment to cover for mistakes.

Programming language: We selected Java as a programming language to ensure that our entire population sample had expertise in it. It is the main teaching language at our university, so our students are all familiar with it. It is entirely possible that other programming languages and libraries would produce different results. Further work is needed to make any kind of generalizable statements.

Study length: We would like to point out the length of the study as another limitation. We picked a whole working day as the length due to the complexity of the task, but refrained from using a multi-day time frame, since it would introduce uncertainty as to what participants did in their off-hours. Acar et al. show that the used information sources have a big effect and we need to be able to capture that [112]. We conducted extensive pilot studies to tailor the tasks to the time available and did not note any fatigue effects. Nonetheless, the study length is sure to have an effect - in particular in comparison to the very short studies carried out in related work [49, 50, 112, 176]. Whether short, one-day, or multi-day studies are more ecologically valid is unknown and must be examined further in future work.

Overall, at present this study should be seen as the first step to offer insights into an extremely complex problem, and the results need to be interpreted as such. Our results need to be replicated and further methodological research is called for. All the above limitations need to be studied and expanded on. In particular, future work should examine different types of developers such as volunteers from online forums, for-hire developers, local developers, etc. and different task descriptions and task settings.

4.7. Summary

It is important to find out why some developers store end-user passwords securely, while others do not. One assumption is that the amount of support the API offers to developers has an impact on secure password storage. However, our study suggests that API usability is not the main factor; in addition to the factor of documentation discovered by Acar et al. [112], task framing and developer misconceptions play a very important role in the (in)security of code created by developers.

We designed a laboratory study with four scenarios in order to analyze the different behaviors and motives of developers when implementing a password-storage task. We selected two popular Web application frameworks with different levels of built-in support for secure password storage, Spring and JSF. Additionally, one group was told that the purpose of the study was to examine the usability of Java frameworks while the other group was explicitly told the study was about security and was directed to store the passwords securely. The key observations from our study are as follows:

Developers think of functionality before security. We conducted semi-structured interviews immediately after the participants finished working on the implementation task. The interviews provided us valuable insights into developers' perceptions of task accomplishment. Just as security is a secondary task for most end users, our developers stated that they always concentrated on the functionality of the task first before (if at all) thinking of security.

Asking for security can make a difference. Our results indicate that there are big differences if a developer is explicitly asked to consider security. None of our non-prompted participants thought that the task of storing passwords needed a secure solution, despite password storage obviously being a security sensitive task. By contrast, participants who were advised to store the end-user password securely used various hash functions with different configurations. The configurations varied in their level of security. Most participants of the prompted Spring group made use of Spring Security's `PasswordEncoder` interface. Thus, they were less likely to make mistakes.

Standards and advice matter. None of the participants who attempted to implement secure password storage produced a solution that met current academic standards. This needs to be seen as a call to action for academics to find better ways to disseminate the latest results in standards and tutorials as well as integrate them with popular frameworks. For the latter option, the trade-off between backward compatibility, flexibility, and security needs to be re-examined.

Opt-out security. At present, even frameworks which have high-level support for secure password storage offer this in an opt-in manner, i.e., developers have to know and want to use this feature. Based on the many misconceptions and lack of primary-task motivation, it would be beneficial to offer safe defaults. This is an open research problem, since it is non-trivial to recognize when developers are writing passwords in plaintext as opposed to other strings across frameworks and IDEs. Ensuring that developers cannot accidentally store a password in plain text or with weak security is a difficult challenge, but one worth tackling.

This study only presents a first qualitative look at password storage as a research problem. In future work, studies are needed to validate and extend the results. Firstly, our student sample limits the generalizability; thus, studies involving programmers with different skills and backgrounds from companies as well as freelancers are needed. Secondly, it is necessary to increase the sample size, in order to gather quantitative data. Thirdly, once the problem is fully understood, work needs to begin on creating more usable password storage solutions to help developers with this critical task.

Based on the findings presented above, further research was conducted in this field; it is described in the following chapters. The student sample size was increased to further examine and validate the findings of the present study. Moreover, further studies on password storage were conducted with freelancers and with programmers employed by companies.

5. A Password-Storage Quantitative Usability Study with CS Students

Disclaimer: *The contents of this chapter were published as part of the publication "Deception Task Design in Developer Password Studies: Exploring a Student Sample" (Fourteenth Symposium on Usable Privacy and Security (SOUPS) 2018) [116]. This paper was produced in cooperation with my co-authors Anastasia Danilova, Christian Tiefenau, and Matthew Smith. For the programming task, I used the same web application frame that I designed and implemented for the study presented in Chapter 4. The results from that study were published in [113]. I designed the survey for this study, and I was given advice by Anastasia Danilova and Matthew Smith. Matthew Smith developed the hypotheses that were examined in this research work. I set up and performed the study in the laboratory. Anastasia Danilova also assisted me with participant recruitment and supervision, and Christian Tiefenau provided technical assistance during the execution of the study. Data analysis was a joint work with Anastasia Danilova and Christian Tiefenau. The participants' survey answers were analyzed by Anastasia Danilova and me. After that, we discussed our findings, which were published in the paper. Moreover, I manually analyzed and systematically categorized all the websites that participants used to copy and paste programming code for their password-storage submissions. Matthew Smith advised me during that categorization. Christian Tiefenau manually analyzed all the participants' video recordings to find out whether they used a search engine to complete the survey. Statistical analysis of the data was a joint work of Anastasia Danilova, Christian Tiefenau, Matthew Smith and me. I discussed key insights and implications of the work with Anastasia Danilova and Matthew Smith. Finally, I prepared the paper for publication in cooperation with Anastasia Danilova and Matthew Smith.*

5.1. Motivation

Applying the philosophy and methods of usable security and privacy research to developers [2] is still a fairly new field of research. As such, the community does not yet have the body of experience concerning study design that it does for end-user studies. Many factors need to be considered when designing experiments. In what setting should they be conducted: a laboratory, online, or in the field? Who should the participants be: computer science students, or professional administrators and developers? Is a longitudinal study needed, or is a first contact study sufficient? Should a qualitative or quantitative approach be taken? How many participants are needed and can realistically be recruited? Is deception necessary to elicit unbiased behavior? How big do tasks need to be? And so forth. All these factors have an influence on the ecological validity of studies with developers. Thus, research is needed to analyze the effects of these design variables.

In this chapter, we present a study exploring two of these design choices. First, we examine the effect of deception/prompting[1] on computer science students in a developer study. To do so, we extended a developer study on password storage (primary study) using different study designs (meta-study) to evaluate the effects of the design.

In end-user studies, deception is a divisive topic. For instance, Haque et al. [202] argue that deception is necessary for password studies: "We did not want to give the participants any clue about our experimental motive because we expected the participants to spontaneously construct new passwords, exactly in the same way as they do in real life." However, Forget et al. [203] explicitly told their participants that they were studying passwords and asked participants to create them as they would in real life, in the hope of getting more realistic passwords. In an experiment to determine whether stating that the study is about passwords has an effect (i.e., priming the participants), Fahl et al. [71] found that there was no significant effect in an end-user study. Thus, there is evidence that deception is not needed for end-user studies. This is particularly relevant in terms of ethical considerations, since deception studies should only be used if absolutely necessary and the potential harm to participants must be weighed carefully.

We face similar questions when designing developer studies. For example, should we inform participants that we are studying the security of their password storage code and thus prompt them, or do we need to use deception to gain insights into their "natural" behavior?

Second, we share our insights on the differences between our quantitative study and a qualitative exploration of password storage. One of the big challenges of developer studies is recruiting enough participants to conduct quantitative research. To examine this, we extended our qualitative password storage study [113] to implement a quantitative analysis and contrast the insights gained with both methods.

The rest of the chapter is structured as follows. In Section 5.2, we introduce our study methodology and explain how the study was extended. Section 5.2.6 contains the main hypotheses of our study and Section 5.3 discusses the ethical considerations. Section 5.4 presents the results and Section 5.5 discusses the methodological contributions. Section 5.7 discusses the limitations of our study. Finally, Section 5.6 summarizes the take-aways and Section 5.8 concludes the chapter.

[1]In [116], the term "priming" was erroneously used to describe security prompting. A detailed description of the difference between security priming and security prompting is provided in Section 2.1.3.

5.2. Methodology

The aim of our study was to gain insight into the design of developer studies. To that end, we used two different kinds of independent variables (IVs). The first was on the meta-level, i.e., variables concerning study design. In our case, we had two meta-IVs: task design (prompting and deception) and type of study (qualitative and quantitative). We refer to these as meta-variables of the meta-study. We also have an independent variable concerning the actual study subject, in our case the framework used to store passwords (JavaServer Faces [JSF] or Spring). We refer to this variable as the primary variable of the primary study.

The study presented in this chapter is an extension of our previous qualitative study on password storage (see Chapter 4) [113]. This quantitative study was planned at the same time to facilitate the analysis of the study type meta-variable, comparing the qualitative and quantitative approaches.

To summarize the qualitative study: participants were told that they should implement the user registration functionality for a social networking platform. Half the participants were instructed to use the Spring framework, which has built-in features for secure password storage. The other half was instructed to use JSF, a framework with manual support for password storage. This part of the design addressed the primary study. Additionally, half the participants were told the purpose of the study was to examine their password behavior and that they should store the passwords securely. The other half received a deceptive study description, which stated that the study was about the usability of APIs. For more detailed information and the exact phrasing of the tasks, see Chapter 4. After the task was completed, a questionnaire was administered and semi-structured interviews were conducted. For the task description and the interviews, participants could choose their preferred language, either English or German. The survey, however, was in English and had to be answered in English. The study was set up for 8 h.

The main difference between the two studies was that in the qualitative study, the exit interviews were used to gain qualitative insights into the development process, while in the quantitative study, we used the survey responses and data gathered by the platform to conduct statistical testing. The hypotheses for this work were developed before the qualitative analysis in Chapter 4 was started. This approach allowed us to gain insights into how a qualitative approach compares to a more quantitative approach.

In the combined study, we examined the following independent variables: for the primary study, the IV was the *framework* used for development (either Spring or JSF). For the meta-study, we used the IVs *prompting* (deception or true purpose) and the *type of study* (qualitative or quantitative).

Participants for both studies were recruited together via a pre-screening survey advertised through the computer science email list of the University of Bonn and flyers on the computer science campus. In total, 82 computer science students completed the questionnaire. Of these, 67 were invited to take part in the study. Seven of these were recruited for pilot studies, leaving 60 invited participants.

The first 20 participants were used for the qualitative study described in Chapter 4 and published

Gender	Male: 77.5%	Female: 15%	Prefer not to say: 7.5%
Ages	mean = 24.89	median = 25	sd = 2.89
Level of Education	Bachelor: 30%	Master: 65%	Other: 5%
Study Program	Computer Science: 82.5%	Media Informatics: 15%	Other: 2.5%
Country of Origin	Germany: 32.5% Iran: 5% Indonesia: 2.5% Finnland: 2.5%	India: 27.5% United States: 2.5% Turkey: 2.5% Uzbekistan: 2.5%	Syria: 5% Korea: 2.5% Pakistan: 2.5% Prefer not to say: 2.5%
Java Experience	< 1 year : 42.5% 6-10 years: 5%	1-2 years: 27.5%	3-5 years: 25%

Table 5.1.: Demographics of 40 participants

in [113]. The remaining participants were used to extend the participant pool for this study. Although we had not planned to do a qualitative analysis on the remaining candidates, we conducted the exit interviews with all participants. This was done to treat all participants equally and to enable extending the qualitative analysis beyond the initially planned 20 in the event we did not reach saturation.

We removed two participants from the dataset of Chapter 4, JN1 and SP2. Due to a technical fault, the code history was not stored for JN1, and SP2 misunderstood the task so completely that no useful data was collected. This was not a big problem for the qualitative analysis but would have made the quantitative comparisons more complicated. Two random participants with a similar skill profile were selected as replacements. Of the remaining 30 invited participants, only 22 showed up. This left us with a total of 40 participants. Participation was compensated with 100 euros. Table 5.1 shows the demographics of all 40 participants. In the rest of the chapter, we will present the quantitative analysis based on these 40 participants. The four conditions being tested are shown in section 5.2.1. In addition, we will contrast the qualitative findings in Chapter 4 with the quantitative findings.

5.2.1. Conditions

We conducted an experiment with 40 computer science students in order to explore whether task framing and different levels of framework support for password storage affect the security of software. Participants were asked to implement a registration process for a web application in a social network context, as described in Chapter 4. The experiment was conducted under the following four conditions (see Section 2.1 for more details):

1. **Prompting** - Participants were explicitly told to store the user passwords securely in the *Introductory Text* and in the *Task Description.*
2. **Non-prompting** - Participants were told the study is about API usability, but were not explicitly asked for *secure* password storage.

3. **Framework with opt-in support for password storage** - Participants were advised to use a framework offering a secure implementation option, which could be used if they thought of it or found it. Spring was chosen as a representative framework.

4. **Framework with manual support for password storage** - Participants were advised to use a framework with the weakest level of support for password storage. Thus, they had to write their own salting and hashing code using just crypto primitives. JSF was considered as a suitable web framework in this case.

Java was selected as the programming language because it is one of the most popular and widely used programming languages for applications and web development [120–125]; in addition, it is regularly taught at our university. Therefore, we reasoned that we would be able to recruit a sufficient sample of computer science students for our study.

Since a related study [49] has shown that self-reported skills of developers affect the study results, we used randomized condition assignments and counterbalanced for participants' skills reported in the pre-questionnaire (this is known as Randomized Block Design [204]). The pre-questionnaire can be found in Appendix A.2.

5.2.2. Deception

We examined the effect that concealing the true purpose of the study had as opposed to openly making it about secure password storage. Kimmel indicated three stages in which deception can be integrated: *subject recruitment, research procedure and post-research/application* [143, p.65]. In our study, we investigated whether participants made sure to store end-user passwords securely, if they were not explicitly told to do so in either the *Introductory Text* or the *Task Description* (non-prompted group). In the recruiting phase, all candidates (prompted and non-prompted) were told the purpose of the study is API usability research (*"The goal of the study is to test the usability of different Java web development APIs."*). Consequently, we used deception in the *subject recruitment* and *research procedure* stages. More details are provided in Section 2.1.3.

5.2.3. Experimental Environment

The experiment was performed in an in-person laboratory, which allowed us to control the study environment and the participants' behavior. We created an instrumented Ubuntu distribution designed for developer studies that included code-specific tracking features. Thus, we were able to collect all data produced by the participants within the 8 h sessions (e.g., the web history and program code history). Every code snippet that was compiled was secured in a history folder. In addition to a video recording of the participants' desktops, the setup also allowed us to take frequent snapshots of their progress. In order to capture copy/paste events, we used *Glipper* [205], a clipboard manager for GNOME, which we modified slightly to meet our requirements (e.g., adding a time stamp to the events in a log file). In this manner, the study environment allowed us to identify all participants who copied and pasted code for password storage and the websites from which they received the code.

5.2.4. Survey

Before working on the task, participants filled out a short entry survey regarding their expectations for task difficulty. They also completed a self-assessment of their programming skills (see Appendix B.1). After finishing the implementation task, participants were required to complete an exit survey (see Appendix B.2). We asked participants for their demographics, security background knowledge, programming experience, and experience with the task and APIs. Furthermore, we asked open questions that could be answered with free text. Two coders independently coded the participants' answers by using Grounded-Theory and compared their final code books using the inter-coder agreement. The Cohen's kappa coefficient (κ) [199] for all themes was 0.78. A value above 0.75 is considered a good level of coding agreement [200].

To analyze the usability of the APIs, we applied the 11-question scale suggested by Acar et al. [50] (Appendix B.2.1), since it is more developer-oriented than the standard System Usability Scale (SUS) [206], which is more end-user oriented. Acar et al.'s usability scale is a combination of the cognitive dimensions framework [207], usability suggestions from Nielsen [208], and developer-related recommendations from Green and Smith [2].

5.2.5. Scoring Code Security

We used the same scoring system as was used in Chapter 4. The description of the scoring system can be found in Section 2.2. For each solution, we examined its **functionality and security**. We rated participants' solutions as functional if "an end user was able to register the Web application, meaning that his/her data provided through the interface was stored to a database" [113].

We used two measures to record the security of a participant's solution. Every solution was rated on a scale from 0 to 7, according to the security scale introduced in Section 2.2.1. This value is referred to as the *security score*. In addition, we used a binary variable called *secure* which was given if participants used at least a hash function in their *final* solutions and thus did not store the passwords in plain text.

We were also interested in participants who attempted to store user passwords securely, but struggled and then deleted their attempts from their solutions (this was coded as *attempted but failed*, or *ABF*). For this, we collected and analyzed participants' code history. In order to identify security attempts, we used the Unix *grep* utility. With grep, we searched for security-relevant terms based on the frameworks and best practices. The following search words were used for security attempt identification: encode, sha, pbkdf2, scrypt, hashpw, salt, MD5, passwordencoder, iterations, pbekeyspec, argon2, bcrypt, messagedigest, crypt. When a term was found, we analyzed the code snippets manually.

It is important to note that we still gave security scores to participants who implemented secure password storage but failed to create a functional solution, i.e., the user registration did not work. The rationale for this was that we were interested in how participants stored passwords. All other parts of the task were distraction tasks and thus of less relevance.

5.2.6. Hypotheses & Statistical Testing

The seven main hypotheses examined for the quantitative analysis are presented in Section 2.3.1. H-P1 and H-P2 concerned the meta-focus, namely, the effect of prompting/deception. H-F1 and H-F2 concerned the A/B test comparing the two frameworks, and H-G1, H-G2, and H-G3 were general tests concerning password storage security. Section 2.3.5 provides information on the statistical analysis.

5.3. Ethics

At the time of the study, our institution did not have an IRB for computer science studies. The study design was instead discussed and cleared with our independent project ethics officer. Our study also complied with the local privacy regulations. Participants gave written informed consent before participating in the study. Since half our participants underwent a deception condition, the study ended with an in-person debriefing, where all participants were informed of the true purpose of the study. Most participants were not bothered by the deception condition at all. However, some participants felt that they were judged unfairly and they stated that they would have included security if we had asked for it. After re-stating that this was completely fine, that we were interested in the APIs' ability to nudge developers toward security, and that they were not at fault, there did not seem to be any lingering negative feelings. There were also positive reactions to the deception. The majority of participants remarked that they learned a lot through the deception and will be more aware of security in future tasks and jobs, even if they are not explicitly asked to think of security.

5.4. Results

While our main goal was analyzing the meta-study results, we began by analyzing the functionality and security of the primary study since these results were needed for the meta-analysis. Sections analyzing one of the seven main hypotheses are marked with the hypothesis label.

5.4.1. Functionality

Here, we discuss the functionality of the code the participants produced. We considered a solution as functional if an end-user account could be created. Figure 5.1 shows the distribution of functional solutions across our conditions. Of all 40 participants, 26 (65%) produced a functional solution. As

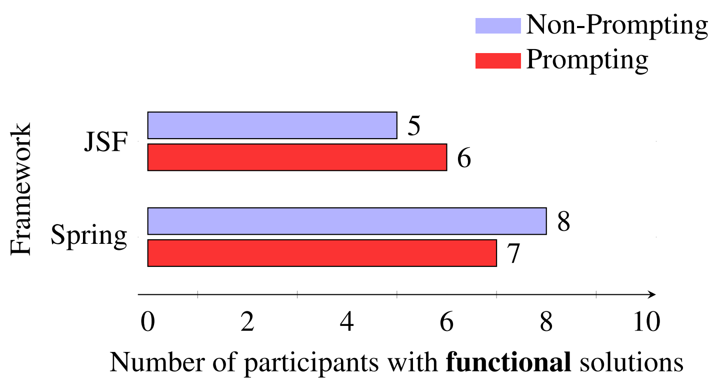


Figure 5.1.: Functionality results per framework, split by prompted vs. non-prompted groups.

shown in Figure 5.1, the number of participants who were able to solve the functional task is a bit higher in the Spring group compared to the JSF group.

Framework effects functional solution (H-F2)

Eleven of 20 (55%) participants using JSF and 15 of 20 (75%) using Spring managed to solve the task functionally. These differences were not statistically significant (sub-sample = all, FET: p = 0.32, odds ratio = 2.40, CI = [0.54, 11.93]). Thus, we do not reject H-F2. However, we only had a power of 0.17, so this effect is worth looking at in follow-up studies. Interestingly, a significant result would mean that the more complex framework actually has better usability with respect to functional solutions.

Part-time job in computer science

We asked our participants whether they had a part-time job in computer science. In prior research, Acar et al. counted students who had part-time jobs as professionals [49]. We, however, found no significant effect between having a part-time job in computer science and a functional solution (sub-sample = all, FET, p = 1.0, odds ratio = 0.84, CI = [0.19, 3.89]).

5.4.2. Security

Figure 5.2 shows the distribution of secure solutions across our conditions. Twelve (30%) of our 40 participants implemented some level of security for their password storage. Of the 20 participants in the non-prompted groups, 0% stored the passwords securely. While we had expected significantly fewer secure solutions in the non-prompted groups, we were surprised by this extreme result. From the prompted group using JSF, 5 of 10 (50%) implemented some level of security (mean security score = 2.15, median = 1, sd = 2.67). From the prompted group with the Spring framework, 7 of 10

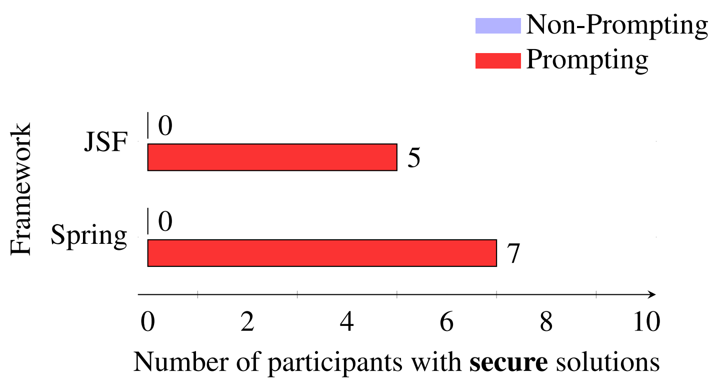


Figure 5.2.: Security results per framework, split by prompted vs. non-prompted groups.

(70%) participants implemented some level of security (mean security score = 4.2, median = 6.0, sd = 2.9). Table 5.2 shows an overview of the security scores achieved by our participants.

More Java experience, more security (H-G1)

In prior research, Acar et al. found that more Python experience leads to more security [49]. We wanted to examine this effect within our sample. We found no significant differences in our study (sub-sample = all, Kruskal-Wallis: $\chi^2 = 4.118$, $p = 0.249$, *cor-p* = 0.498, family = 6). Between the different groups of Java experience, the security score showed no significant effect. However, it must be mentioned that Acar et al. studied student and professional volunteers recruited on GitHub without compensation. Their participants had a wide range of years of experience in Python compared to our students in Java. So we have a number of differences in the samples. It is important to note this difference since it makes sense to take skills into account during condition assignment in randomized control trials. In short, we failed to confirm H-G1.

Previous password experience (H-G2)

We hypothesized that participants who had previous experience storing user passwords in a database backend would be more likely to add security in the study. Therefore, we wanted to test whether participants who reported having stored passwords before performed differently regarding security compared to participants who had never stored passwords before. Nine non-prompted and 15 prompted participants reported having stored passwords prior to the study. We found no significant differences in security in comparing the different groups of participants (sub-sample = all, FET: p = 0.297, *cor-p* = 0.498, odds ratio = 2.54, C.I = [0.49, 17.72], family = 6). We thus fail to reject the null hypothesis of H-G2 and cannot draw conclusions on this hypothesis. Furthermore, we calculated a power of 0.19, indicating that the effect is not reliable.

	Time	**Functionality**	**Security**					
			Hashing			**Salt**		
	(hh:mm)	Storage working	Hashing function (at most +2)	Digest size (bits) (+1 if ≥ 160)	Iteration count (at most +1)	Generation (at most +2)	Length (bits) (+1 if ≥ 32)	**Total** (7)
JN2*	04:05	✓						
JN3*	03:01	✓						
JN4*	04:11	✗						
JN5*	05:30	✗						
JN6	05:13	✓						
JN7	07:33	✗						
JN8	07:33	✗						
JN9	06:08	✓	SHA1	160	1	end-user email address	8	3 (ABF)
JN10	03:45	✓						
JN11	06:36	✗						
JP1*	04:55	✓						
JP2*	03:12	✓	PBKDF2(SHA256)	512	1000	SecureRandom	256	5.5
JP3*	05:29	✓	SHA256	256	1			2
JP4*	04:12	✓	PBKDF2(SHA1)	160	20000	SecureRandom	64	6
JP5*	06:32	✓						
JP6	07:33	✗						
JP7	06:08	✗	BCrypt	184	2^{12}	SecureRandom	128	6
JP8	07:22	✗						
JP9	07:18	✗	BCrypt	184	2^{8}	pgcrypto	128	5 (ABF)
JP10	04:45	✓	SHA256	256	1			2
SN1*	03:15	✓						
SN2*	02:24	✓						
SN3*	02:01	✓						
SN4*	04:01	✓	BCrypt	184	2^{10}	SecureRandom	128	6 (ABF)
SN5*	04:50	✓						
SN6	07:03	✗						
SN7	05:35	✓						
SN8	07:33	✗						
SN9	05:31	✓						
SN10	03:23	✓						
SP1*	03:15	✓	BCrypt	184	2^{10}	SecureRandom	128	6
SP3*	07:00	✗	BCrypt	184	2^{10}	SecureRandom	128	6
SP4*	03:39	✓	BCrypt	184	2^{10}	SecureRandom	128	6
SP5*	03:44	✗						
SP6	07:33	✓						
SP7	01:49	✓	BCrypt	184	2^{11}	SecureRandom	128	6
SP8	05:59	✗	#					0 (ABF)
SP9	05:50	✓	BCrypt	184	2^{10}	SecureRandom	128	6
SP10	05:53	✓	BCrypt	184	2^{10}	SecureRandom	128	6
SP11	03:15	✓	BCrypt	184	2^{10}	SecureRandom	128	6

Labeling of participants: S = Spring, J = JSF, P = Prompting, N = Non-Prompting
* = Used for the qualitative study presented in Chapter 4
# = Used Spring Security's `PasswordEncoder` interface without deciding for an algorithm

Table 5.2.: Password security evaluation, including participants who attempted to implement security but failed (ABF).

Framework effects security score (H-F1)

In this section, we only consider those participants who attempted security. We wanted to examine whether the framework used affected the security score (including ABF scores). We expected that Spring might score better because, in contrast to JSF, it offers built-in functions for storing passwords securely by using hashing, salting and iterations.

The descriptive statistics for the JSF group are Min 2, Median 5.5, Mean 4.3, and Max 6. The descriptive statistics for the Spring group are Min 6, Median 6, Mean 6, and Max 6. Due to the Bonferroni-Holm correction, the difference between the two groups is not flagged as significant (sub-sample = all $\wedge$ attempted security = 1 , Mann-Whitney U = 15, p = 0.051, *cor-p* = 0.20, family = 6). It does seem likely, though, that a larger sample would confirm the trend that Spring participants earned higher scores than JSF participants. This will be put into further context in section 5.4.4.

Usability of frameworks

We used the usability score from Acar et al. [50] (see Appendix B.2.1) to evaluate how participants perceived the usability of the two frameworks. We compared the values of the usability score for all four groups: non-prompted JSF (mean = 48.25, median = 51.25, sd = 13.54), prompted JSF (mean = 50.50, median = 53.75, sd = 9.78), non-prompted Spring (mean = 50.50, median = 55.00, sd = 20.10), prompted Spring (mean = 58.75, median = 57.50, sd = 15.65). We found no significant effect comparing all four groups (sub-sample = all, Kruskal-Wallis: χ^2 = 3.169, p = 0.37). Furthermore, we examined whether the frameworks had different usability scores when the participants attempted to solve the task securely. We did not find a significant effect in this case either (sub-sample = all $\wedge$ attempted security = 1, Mann-Whitney U = 21.5, p = 0.29).

Security awareness

Fourteen prompted and two non-prompted participants believed that they managed to store user passwords securely. The two non-prompted participants erroneously believed they had stored passwords securely (JN5, JN7). Their given survey answers suggested that neither had any background knowledge of password storage security at all. The prompted participants were additionally asked whether they would have been aware of security if we had not explicitly ask them for it. Nine of the 14 participants indicated they would have stored user passwords securely, even if they had not been explicitly asked to do so. The fact that only two out of 20 non-prompted participants attempted security suggests this is overly optimistic.

Security classes

In prior work, Acar et al. found that security courses had a significant effect on security [49]; therefore, we asked our participants which courses they had attended at our university in the past. We gave one point per security-relevant course. Since not all Masters students had completed their undergraduate studies at the same university, we also asked for other courses. None of the participants added a security-relevant class in the open question space. Participants reported they had attended between 0 and 4 security classes (mean = 0.8, median = 1, sd = 0.99). We found no significant evidence in the overall group (sub-sample = all, FET: p = 0.737, odds ratio = 1.39, CI = [0.29, 7.022]).

Part-time job in computer science

We found no effect between having a part-time job in computer science and a secure solution (sub-sample = all, FET, p = 1.0, odds ratio = 1.10, CI = [0.22, 5.30]).

Web browser history & task completion time

In order to analyze the web browser history, we aggregated all our participants' browser history. We assessed the visit count of all participants (mean = 179.0, median = 174.5, sd = 97.02). We could not analyze the browser history of one of our participants, because he had deleted it after completing the task. We found a total of 6224 distinct web pages for all participants. We also measured the time our participants needed to solve the task [hours] (mean = 5.11, median = 5.35, sd = 1.72). On average, participants visited 36.2 pages per hour (mean = 36.2, median = 31.52, sd = 16.44). We tested whether there was a difference in security that depended on the number of websites participants used. The results show that the website count was not significantly relevant (logistic regression, odds ratio = 1.0 , C.I = [0.99, 1.00], p = 0.423).

5.4.3. Prompting

Prompting leads to more attempts to store user passwords securely (H-P1)

The main goal of our study was to measure the effect of prompting. Only two of 20 non-prompted participants attempted to store the passwords securely, compared to 14 of 20 in the prompted groups. This difference is statistically significant (sub-sample = all, FET: p = 0.000*, *cor-p* = 0.001*, odds ratio = 19.02, C.I = [3.10, 219.79], family = 6). Thus, we can reject the null of H-P1 and conclude that prompting has a significant effect. We already stated we were surprised that no non-prompted participant achieved a secure solution. This is mirrored in the very low number of participants who attempted to create a solution. However, we were also surprised that six participants in the prompted group did not attempt a secure solution, since it was explicitly asked of them. Of these, though, three also did not manage to create a functional solution. In the exit survey, all six participants stated that they had not achieved an optimal solution and cited technical difficulties that prevented them from attempting to create a secure solution. For instance, SP6 noted: "[I] encountered errors in connecting with the DB through Spring JPA and was not able to come up with the solution. As a result [I] could not focus on implementing an algorithm to securely store the password."

It is interesting to note that even when security was explicitly stated as the goal of the study, these participants still wanted to create the functional solution before adding the security code.

Prompting effect on achieving a secure solution once the attempt is made (H-P2)

We had hypothesized that the prompting effect would only influence whether a participant would think of adding security, but once a participant had made the decision to add security, the will to follow through would be independent of prompting. Now, it is very difficult to make a convincing case of no-effect using frequentist statistics with a small sample size; however, this may not be a concern. It turned out that there might actually be an effect. In the non-prompted group, two of 20 attempted security but did not follow through to achieve a secure solution. In the prompting group, 14 of 20 attempted and 12 achieved a secure solution. The difference between the groups is significant before correcting for multiple testing (sub-sample = all $\wedge$ attempted security = 1, FET: *p* = 0.05, *cor-p* = 0.20, odds ratio = Inf, CI = [0.64, Inf], family = 6). The same goes for the security scores (sub-sample = all $\wedge$ attempted security = 1, Mann-Whitney U: 2.0, *p* = 0.034*). Although this effect was not significant after correction, we think this is an important observation which should be examined in future studies. While it is possible that the small number of attempts in the non-prompted group skewed our results, it is also possible that the failure to mention security in the task not only meant participants were not explicitly informed that security is important for password storage, but potentially discouraged participants who knew this from implementing it. This could have implications outside of study design since this effect is likely to occur in everyday life as well where developers might not be explicitly asked to secure their code and thus be dissuaded from doing so even if they know they should.

While we fail to reject the null of H-P2 due to the Bonferroni-Holm correction, we find the data to be highly interesting and suggest examining this effect in future studies.

5.4.4. Copy/Paste

Security and copy/paste (H-G3)

Our analysis of the copy/paste behavior of our participants showed another interesting result.

Of the 40 participants, only 17 copied and pasted code. Of these, 12 created a secure solution. The surprising aspect is that all secure solutions come from participants who copied and pasted security code. Not a single "non-copy/paste" participant achieved security. This difference was statistically significant (sub-sample = all, Mann-Whitney U = 57.5 *p* = 0.000*, *cor-p* = 0.000*, family = 6). Thus, we reject the null of H-G3. However, it is noteworthy that we see a positive effect of copy/paste. This is in contrast to previous work by Acar et al. [112] and Fischer et al [186]. For example, Acar et al. stated in their discussion: “Because Stack Overflow contains many insecure answers, Android developers who rely on this resource are likely to create less secure code” [112]. And Fisher et al. stated in their conclusion: “We show that 196,403 (15%) of the 1.3 million Android applications contain vulnerable code snippets that were very likely copied from Stack Overflow” [186].

These negative views are in stark contrast to our findings that 0% of participants who did not use copy/paste created a secure solution. We do not dispute the findings of Acar et al. and Fisher et al., but we do show that there is also a significant positive effect of copy/paste.

This finding also changes how we must interpret the difference in security scores between the two framework conditions presented in section 5.4.2. All secure Spring participants scored 6 points, while the JSF scores varied between 2 and 6. This could indicate that the Spring API has better usability, because it has safer defaults. However, this usability advantage seems to only affect our participants indirectly, via the web sources they use. This suggests that it is worth considering testing the usability of APIs not only with software developers but also with those who create web content. In the following section, we take a closer look at the websites used by our participants.

Websites used for copy/paste

Almost half of the participants (42.5%; 17/40) copied password storage examples from various websites on the Internet and pasted it to their program code. Of these, 82% (14/17) were prompted participants. In all other cases, participants copied code from websites covering storage of user data in general (e.g., name, gender, email), adapting it for passwords. These websites were not considered for further analysis, since we were only interested in password storage examples.

Table 5.3 shows all websites from which participants copied and pasted code for password storage into their solutions. The table also considers participants who attempted to store user passwords securely but did not include the security code in their final solutions (ABF). We manually analyzed all proposed examples for password storage on these websites by using the same security scale as applied to the evaluation of participants' code (see Section 2.2.1). If websites introduced generic solutions without predefined parameters for secure password storage, but discussed how these should be chosen in order to achieve security (e.g., OWASP: General Hashing Example), we still awarded points for these parameters according to the security scale. Additionally, we compared the security scores participants received for their solutions with the scores of password storage examples from the websites they used. Since websites often contain more than one code snippet, we manually scored all of them and then used the following classification of snippets:

- **Most insecure example** - The worst solution we found on the page.
- **Obvious example** - The most obvious solution in our subjective assessment, e.g., answers on Stack Overflow that are rated with a high score by the community. For all other websites, we classified examples as obvious if they were posted at the beginning of the website.
- **Most secure example** - The solution with the highest security score.

We found that all participants who implemented password storage security (100%, 12/12) copied their program code from websites on the Internet. The majority, 75% (9/12) of participants, achieved the almost maximum score of 6/7 points in our study. These participants copied and pasted code from websites introducing up-to-date, strong algorithms. One thing the websites had in common was that all solutions had good security scores. Only one participant was on a website where the least secure example was "only" a 5.5 score. However, the most obvious example was scored with a 6 and taken by the participant.

Participant	Security score	Website	Description	Most insecure example	Obvious example	Most secure example
JN9	3 (ABF)	www.sha1-online.com/sha1-java/	Blog Post: SHA1 Java	2	2	2
JP2	5.5	https://www.owasp.org/index.php/Hashing_Java	OWASP: General Hashing Example	6	6	6
		https://stackoverflow.com/questions/18268502/how-to-generate-salt-value-in-java	Stack Overflow: Salt Example	3	3	3
		https://howtodoinjava.com/security/how-to-generate-secure-password-hash-md5-sha-pbkdf2-bcrypt-examples/#PBKDF2WithHmacSHA1	Blog Post: Hashing Example	1	1	7
JP3	2	www.mkyong.com/java/java-sha-hashing-example/	Blog Post: Hashing Example	2	2	2
JP4	6	http://blog.jerryorr.com/2012/05/secure-password-storage-lots-of-donts.html	Blog Post: Hashing Example	5.5	6	6
JP7	6	http://javaandj2eetutor.blogspot.de/2014/01/jsf-login-and-register-application.html	Blog Post: Hashing Example	0	0	0
		www.mindrot.org/projects/jBCrypt/	Documentation: Java Implementation jBCrypt	6	6	6
JP9	5.5 (ABF)	https://www.meetspaceapp.com/2016/04/12/passwords-postgresql-pgcrypto.html	Blog Post: Hashed Passwords with PostgreSQL's pgcrypto	5.5	5.5	5.5
JP10	2	https://stackoverflow.com/questions/33085493/hash-a-password-with-sha-512-in-java	Stack Overflow: Hashing Example	3	3	5
		https://stackoverflow.com/questions/3103652/hash-string-via-sha-256-in-java	Stack Overflow: Hashing Example	2	2	2
		https://docs.oracle.com/javase/7/docs/api/java/security/MessageDigest.html	Documentation: Class MessageDigest	1	2	2
		https://stackoverflow.com/questions/11665360/convert-md5-into-string-in-java	Stack Overflow: Convert MD5 into String in Java	1	1	1
SN4	6 (ABF)	http://websystique.com/spring-security/spring-security-4-password-encoder-bcrypt-example-with-hibernate/	Blog Post: Hashing Example	6	6	6
SN8	0	https://dzone.com/articles/spring-mvc-example-for-user-registration-and-login-1	Blog Post: Hashing Example	0	0	0
SP1	6	https://stackoverflow.com/questions/25844419/spring-bcryptpasswordencoder-generate-different-password-for-same-input	Stack Overflow: Hashing Example	6	6	6
SP3, SP4, SP11	6	www.mkyong.com/spring-security/spring-security-password-hashing-example/	Blog Post: Hashing Example	6	6	6
SP7	6	https://hellokoding.com/registration-and-login-example-with-spring-xml-configuration-maven-jsp-and-mysql/	Blog Post: Hashing Example	6	6	6
SP8	0 (ABF)	www.websystique.com/springmvc/spring-mvc-4-and-spring-security-4-integration-example/	Blog Post: Hashing Example	6	6	6
SP9	6	https://stackoverflow.com/questions/18653294/how-to-correctly-encode-password-using-shapasswordencoder	Stack Overflow: Hashing Example	5.5	6	6
SP10	6	https://stackoverflow.com/questions/42431208/password-encryption-in-spring-mvc	Stack Overflow: Password Encryption in Spring MVC	6	6	6

Table 5.3.: Websites from which participants copied and pasted code for password storage.

The other three participants came across blog posts and tutorials with outdated or unsecure implementation (JP2, JP3, and JP10). For instance, JP2 copied code from a tutorial that was published in 2013 (Blog Post: Hashing Example). Thus, he adopted an iteration count of 1 000 for PBKDF2, although 10 000 iterations are recommended by NIST today [97]. Interestingly, this tutorial also discussed the usage of `MD5`, `bcrypt`, and even `scrypt` with associated program code examples. The example for `MD5` was listed at the top of the website; we therefore classified it as the *obvious* example. But the author did state that this solution is vulnerable to diverse attacks and should be used with a salt. The blog post also discussed a program example for `scrypt`, which we classified as *most secure*. This was the only website visited by our participants where an example scored 7/7 points. However, JP2 decided to use `PBKDF2`, for which he found a general hashing example at the Open Web Application Security Project (OWASP) website (OWASP: General Hashing Example). Although properties of parameters are discussed on the OWASP website in general, they are not applied in the code example. Therefore, JP2 searched for a similar implementation with predefined parameters and ended up with an outdated iteration count.

JP3 copied code that only contained a weak SHA1-based example. More interestingly, JP10 merged program code from four websites. Although one website included code with three points for an *obvious* example and five points for a *most-secure* example, he received only two points for his final solution. He did not use a salt, despite the fact that he copied code from an *obvious* example on Stack Overflow that considered a function with a salt as an input parameter. However, the example did not include a predefined implementation of the salt and was not implemented by our participant.

An interesting prompting effect can be seen between the two participants, JP7 and SN8, who both copied code from websites in which user credentials were stored in plain text. The prompted participant, JP7, used the unsecure blog post for gaining a functional solution and afterward installed a Java implementation of OpenBSD's Blowfish password hashing scheme, jBCrypt, and received six points. In contrast, the non-prompted participant, SN8, did not take any further action to implement security.

Only two of the 20 non-prompted participants considered security while programming, though they did not provide secure solutions in the end (JN9, SN4). JN9 was able to implement a functional solution storing user passwords securely. However, he accidentally deleted parts of his code, resulting in errors he was unable to correct. At the end, he provided a functional solution without including secure password storage. In terms of copy/paste, JN9 is interesting since the solution he implemented had, at one point, a security score of 3, although the website he used for copy/paste was scored with 2 points. He was the only participant who used a salt that he did not copy and paste from a website, but rather included it by himself. However, he used the user's email address as the salt, which is not considered a security best practice.

SN4 and SP8 both used the same blog post domain introducing an extensive example of how to securely store user credentials to a database backend using Spring. However, Spring is rather feature-rich. For SN4 and SP8 the details of the example were too extensive and too complex so

H	**Sub-sample**	**IV**	**DV**	**Test**	**O.R.**	**C.I.**	***p*-value**	***cor* − *p*-value**
H-P1	-	Prompting	Attempted security	FET	19.02	[3.10, 219.79]	0.000^*	0.001^*
H-P2	Attempted security = 1	Prompting	Secure	FET	Inf	[0.64, Inf]	0.05^*	0.20
H-F1	Attempted security = 1	Framework	Security score (incl. ABF)	Mann-Whitney	-	-	0.051^*	0.20
H-F2	-	Framework	Functional	FET	2.40	[0.54, 11.93]	0.32	-
H-G1	-	Java experience	Security score	Kruskal-Wallis	-	-	0.249	0.498
H-G2	-	Stored passwords before	Secure	FET	2.54	[0.49, 17.72]	0.297	0.498
H-G3	-	Copy/Paste	Security score	Mann-Whitney	-	-	0.000^*	0.000^*

Independent variable, DV: Dependent variable, O.R.: Odds ratio, C.I.: Confidence interval
Corrected with Bonferroni-Holm correction, except for H-F2
Significant tests are marked with *

Table 5.4.: Summary of main hypotheses

that they were not able to comprehend the general idea of secure password storage and thus failed to implement it. For instance, SP8 only added an interface of Spring Security's `PasswordEncoder` to her code without choosing a hashing function at all.

Further, the prompted participant JP9 attempted to store user passwords securely. JP9 used the module *pgcrypto*, which provides cryptographic functions for PostgreSQL. However, the blog post she used was not adapted to the study context. Therefore, she failed to integrate it to her code. Finally, she removed the security attempt completely from her solution.

In summary, no participant who copied/pasted code used the *most unsecure example* on websites. Whenever the *obvious* security score differed from the *most secure* examples (true for 3/21 websites), participants used the latter. If participants' code was merged from more than one website (JP2, JP7, and JP10), participants' security score was always higher compared to the lowest-scored website, considering *most secure examples*.

5.4.5. Statistical Testing Summary

Table 5.4 gives an overview of the seven main hypotheses and the results of our statistical tests, with both the original and Bonferroni-Holm corrected p-values. We have two very clear results. First, concerning the meta-study: prompting has a huge effect. Second, concerning the primary study: copy/paste has a strong positive effect on code security.

The effects of H-P2 and H-F1 were not statistically significant after correcting for multiple testing, but seem promising enough to examine in future work. It is also noteworthy that we did not find a significant effect for H-G1, which has been found in other studies. This is likely due to the fact that with only a student sample, the range of experience was so small that the effect is not large enough. This is important to know since it simplifies study design for developer studies conducted with students.

5.4.6. Examining Survey Open Questions

We analyzed open questions of the exit survey for trends rather than for statistical significance, to gather deeper insights into the rationale behind participants' behavior.

Before mentioning security at all, we asked our participants whether they solved the task in an optimal way (see Appendix B.2, Q2). Thus, we were able to observe whether non-prompted participants based their answers on functionality rather than on security. Seven out of 40 participants believed their solution was optimal (JP3, JN10, SN1, SN2, SN5, SN7, and SP1). In fact, most of the participants were non-prompted and solved the task functionally but not securely. Some even stated that all requirements were functionally solved and thus their solution was optimal (JN10, SN7, SN1, SN2, and SN5). SN1, for instance, noted: "My [manually performed] tests [...] worked as expected, I should have covered everything." His answer shows that he invested some time in testing his implementation. Still, since SN1 did not think about storing the user credentials securely, it might be interesting to involve security in the testing process as well. The prompted participant SP1, though, argued that his solution was optimal because the security part was sufficiently solved: "It uses `bcrypt` [with the] highest vote on [Stack Overflow link]." In contrast, a number of participants said that the quality of their code was not optimal because it did not rely on best practices, e.g., SP11: "I have probably not used best practices for Spring/Hibernate as it is the first time I used them." Other participants mentioned that exceptions and warnings need to be caught and the code can be written more cleanly and clearly (SP4, SP9, SP10, SN4, and SN9).

If participants believed they stored the user password securely, they were asked whether they solved the task in an optimal way with regard to security (see Appendix B.2, Q9). Only 7 of 40 participants believed that their security code was optimal (JN5, JN7, JP4, JP7, JP10, SP7, and SP11). SP7, for instance, noted that he used an "industry standard way of storing passwords" and assumed that his solution was therefore optimal. While JP3 and SP1 indicated they solved the task in an optimal way at first, they changed their minds when the question was asked in terms of security. While JP3 noted "everything is implemented", thus indicating his solution was optimal, he changed his mind with regard to security, "because the [iteration count] is not implemented yet." SP1 listed three reasons explaining why his solution is not optimal in terms of security: (1) "User is not enforced to use symbol, combination of numbers, etc.," (2) "Storing the password securely does not mean that one [person] cannot hack into another's account," and (3) "Lacking [...] 2 step validation (by phone, for example)." First, SP1 assumed that security should be implemented involving the end user. This assumption was also made by other participants, who noticed that password validation for the end user was missing in their solutions (SN1, SN2, SN4, SP5, JP7, SP9, and SP11). Second, SP1 did not trust password security at all, although he suggested a method for improvement (two-factor authentication). Interestingly, the non-prompted participants, JN5 and JN7, indicated they stored the user password securely in an optimal way. However, we did not find any evidence of security at all, in either their solutions or in their attempts. Their answers suggested a general lack of knowledge of password storage security.

5.5. Methodological Contributions

5.5.1. Deception

While Fahl et al. [71] found no significant difference in password studies in the behavior of end users who were primed that the study was about passwords or received deceptive treatment, we see a very strong prompting effect on the behavior of developers. Both design choices offer interesting insights into the problem of storing passwords securely.

If researchers wish to study the usability of a security API, prompting participants is clearly the best choice, since the majority of participants in the non-prompted group had no contact with the API at all and thus do not produce any data to analyze. The majority of developer user studies fall into this category.

However, these studies only look at one aspect of a much larger problem. In [167] Fahl et al. analyzed the misuse of transport layer security (TLS) APIs in Android. They found that 17% of applications using HTTPS contained dangerous code. However, 53.8% of apps did not use the TLS API at all, exposing a wealth of data to the Internet without any protection. We think it is important to study this aspect as well, and help developers become aware they need to think about security. Our results suggest that deception in studies is a promising way of studying this. It can be argued that the students simply did not include secure storage because they were in a study environment. Some participants even stated this in the exit survey and interviews. However, since there are many cases in the real world in which security is not explicitly stipulated, we think that the non-prompting condition can be a valuable design for studies. This is definitively an area in which more research is needed before a reliable statement can be made.

For now, we do suggest that the usable security community also conducts developer studies using deception instead of focusing only on API use on its own. It is, however, important to conduct a full debriefing at the end to ensure the well-being of participants. In our case, we did not see any issues with the debriefing that were not addressed to the satisfaction of the participants.

5.5.2. Task Length

The most difficult aspect of designing a deception study for developers is that distraction tasks are necessary to avoid tipping off the participants. While most prompting developer studies only take an hour or two, deception tasks require a lot more time making the already difficult task of recruiting participants even harder.

Short tasks

Most related studies are very short [49, 50, 112, 176]. As noticed by Acar et al. [49], tasks for uncompensated developers should be designed in a way that *"participants would be likely to complete them before losing interest, but still complex enough to be interesting and allow for some mistakes."*

Acar et al. [49] conducted an online experiment with 307 uncompensated GitHub users, who were asked to complete three different tasks: (1) URL shortener, (2) credential storage, and (3) string encryption. Each participant was assigned the tasks in random order. For the user credential storage task, only one function was given, which had to be completed by developers. The task was formulated in a straight forward way and it was clear where to insert the needed code and why. Additionally, clear instructions were given to the participants, answering the question when the problem was solved. The participants were not explicitly asked to consider security. In their study, only a small number, 17.4%, stored the user passwords in plain text. A direct comparison cannot be made since the GitHub users were more experienced than the students in our study; however, the short task time and the direct instruction to store the passwords is likely to have an effect as well.

One-day time frame

In contrast to tasks completed over a short time frame, longer studies are more realistic since developers have long-lasting projects and tasks they work on in the real world. In particular, it is possible to create competing requirements, pitting functionality against security in a way that is not possible in short, focused tasks. In Chapter 4, we discussed the design process of the task used in this chapter in detail and how the 8 h time frame was calibrated with several pilot studies. The rational was that 8 hours is the longest time we could reasonably ask participants to remain in a lab setting. In addition, there are a number of benefits to having the participants in a controlled environment. In particular, we could fully configure the lab computers to gather a wealth of information, including full-screen capture, history of all code, copy/paste events, search history, and websites visited. Remote studies could easily use web-based editors to capture code and copy/paste events; however, gathering the rest of the information would be much more intrusive.

Multi-day time frame

In a one-day time frame, we were able to conduct a task that was sufficiently long and complex that participants could perceive security as a secondary task. A multi-day time frame also offers this benefit. For instance, Bau et al. [139] investigated web application vulnerability in a multi-day experiment with eight freelancers. They were asked to develop an identity site for youth sports photo-sharing with login and different permission levels for coaches, parents, and administrators. The freelancers were prompted for security by mentioning that the website "was mandated by 'legal regulations' to be 'secure', due to hosting photos of minors as well as storing sensitive contact information" [139]. The developers promised a delivery period of 35 days. Participants were compensated from three different price ranges (< \$1 000, \$1 000 - \$2 500, and > \$2 500). Two of the eight freelancers stored passwords in plain text, showing a similar distribution as in our prompting condition. This design offers higher ecological validity; however, far less detailed information about the code creation process can be gathered. Both our study and the studies conducted by Acar et

Group		Search	Security search
Prompted	Non-Secure (6)	3	1
	ABF (2)	1	1
	Secure (12)	8	7
Non-Prompted	Non-Secure (16)	7	4
	ABF (2)	1	1

Table 5.5.: Number of participants who searched on the Internet in order to fill out the survey.

al. [112] have shown that information sources play a vital role in code security, which is much trickier to gather in this kind of study. So there is a trade-off between ecological validity and the ability to gather high-fidelity data.

In short, we see benefits in all three time frames and researchers now have initial data to help choose which is most appropriate for their setting.

5.5.3. Laboratory Setting

Many developer studies are conducted online due to the difficulty of recruiting enough participants to come to a lab study. However, we found the information gathered by our instrument OS very valuable. Most developer studies contain both coding tasks and questionnaires. The questionnaires are used both for pre-screening and for gathering information on the task. While it is possible to detect the use of web sources indirectly through paste events, it is also critical to be able to detect the use of online sources during the administration of surveys.

We manually analyzed all the screen capture videos of our participants while they were answering the surveys. We could only analyze the videos of 38 participants due to technical difficulties, which meant that we were missing two videos (JN8 and SN5).[2]

We found that half the participants (20/38) used Google when answering the survey, either searching for framework-related topics (6/38) or for password storage-related topics (14/38; see Table 5.5). Interestingly, half the non-prompted participants who did not attempt to store user passwords securely (4/8) started to search how this could be done while answering the survey. SN1, for instance, copied a survey answer from Wikipedia, explaining what hashing functions are defending against.

Of the prompted participants with secure solutions, 58% (7/12) searched for additional password storage security details, e.g., in order to explain why the used algorithms were optimal or not.

Since our laboratory setting captured this information, we could take it into account during data analysis. In most online settings, this information is not available and thus there can be no certainty that the answers reflect the knowledge of the participant or just their ability to use Google.

This is particularly critical in the use of pre-screening surveys, as is done in most studies (including

[2]We later discovered there was a keyboard shortcut that participants seemed to have used by accident which stopped the recording.

this one). It is common to try to screen out unsuitable candidates who do not have the technical skills needed to take part. Luckily, we only used self-assessment and reported experience to conduct the counter-balancing. However, there are also expert studies which used content-based questions for participant selection, such as the study by Krombholz et al. [43]. Here, the researchers had to be aware that a potentially large number of the participants used Google to answer the questions, which might not properly reflect their actual skills.

Being able to see all searches and information sources in direct relation to questions and answers was very valuable and is an important strength of lab-based studies. We will be releasing the study OS as an open source project, so other studies can easily capture the same information.

5.5.4. Qualitative vs. Quantitative Study Design

Finally, we want to share some observations contrasting the qualitative approach from [113] (Chapter 4) with our quantitative extension. Here, we need to distinguish between the primary study and the meta-study.

Concerning the meta-variable prompting, the qualitative study already delivered a good indication that there was a significant effect, with 0 of 10 non-prompted participants and 7 out of 10 prompted participants achieving a secure solution. However, since small samples tend to produce more extreme results, we would not have recommended basing study design decisions on these results. With a sample size of 40 participants in the present study, we are confident this is not a fluke and that the use of deception changes the behavior of participants dramatically. It would be useful to conduct even larger studies since we currently can only expect to find large effects. However, with regard to study design, we would very much want to catch medium or even small effects as well.

For the primary study, extending the sample size allowed us to conduct an A/B test to compare two frameworks. While H-F1 was not significant in this study due to the addition of the meta-variables and consequent correction for multiple tests, even the relatively small sample size in a normal developer study would be sufficient to get good results. That being said, the qualitative study already highlighted many of the problems faced by developers, and the interviews were very valuable in gaining deeper insights. We did not find much to add to the conclusions of the primary study of [113] (Chapter 4) other than having stronger evidence that library support as offered by Spring has tangible benefits.

A particularly salient benefit to qualitative developer studies is that fewer participants are needed. As such, unless rigorous evidence in the context of an A/B test is needed, we think that usable security research into developers is at a stage where qualitative studies have a lot to offer and encourage the community to be more accepting of them.

5.6. Take-Aways

Below, we summarize the main take-aways from our study.

- Task design has a huge effect on participant behavior and deception studies seem to be a promising method for examining a previously overlooked component of developer behavior when using student participants. That said, we must reiterate important limitations to this finding. We cannot make any claims concerning studies with professionals. It seems likely that even within a group comprising professionals, there will be multiple sub-groups that will react differently under prompting. This will need to be examined in future work. It is also possible that a large portion of this effect is a study artifact. In any case, we recommend more experimentation concerning the design of developer studies. Currently, researchers base task and study design mostly on gut feelings. Since we have shown that one gets vastly different outcomes, we believe it is worth investing the effort into testing multiple designs in pilot studies instead of just going with one design as is currently often the case. We also believe more effort needs to be invested in understanding what motivates developers to implement security instead of focusing too narrowly on the easier measure of API usability.

- The use of Google by participants during surveys is problematic and researchers should not rely on answers reflecting the internal knowledge of the participants. This is particularly relevant for pre-screening surveys and we strongly recommend avoiding use of answers that can be googled for participant selection or condition assignment. If at all possible, we recommend that search behavior and web usage should be tracked, because a) thus, researchers can distinguish between internal knowledge and the ability to search for knowledge; and b) seeing when and what participants google is very enlightening in itself and a valuable research instrument.

- It is our belief that qualitative research into developer behavior offers a good cost/benefit trade-off and that many valuable insights can be gained without the need for large(r) sample sizes. In addition, the use of interviews as opposed to surveys avoids the googling problem. We hope that our comparison of quantitative and qualitative examination of the same topic encourages more qualitative studies and lowers the barriers to entering into this field, since recruitment of participants is one of the biggest challenges.

- While Acar et al. have found that programming language experience has a significant effect on the security of code produced in developer studies [49], we did not find a significant effect for this. In contrast to their study, our student sample had a much smaller range of programming skills; this could explain the lack of a measurable effect. This suggests that it might not be necessary to balance programming experience when working with students, thus simplifying random condition assignment. However, our power on this test was low so this result should be replicated before it is used confidently.

- We found copy/paste has a significant positive effect on the security of our participants' code. The way previous work was set up meant that they mainly found negative effects, thus potentially skewing the perception. We think highlighting the positive side of copy/paste behavior is important.

5.7. Limitations

Our study has several limitations that need to be considered when interpreting the results.

The most noteworthy limitation is that we recruited a convenience sample comprising 40 computer science students from a single university. Despite having a pool of 1600 computer science students at our university and offering fairly high compensation, we did not get more volunteers. We will discuss this limitation in the context of both the primary study and the meta-study. For the meta-study, we wanted a homogeneous sample so we could attribute any changes in outcome to the difference in task design. The limitation of this decision is that our results are not currently transferable to other participant groups. While we believe it is likely that other student samples will produce similar results, we expect bigger differences when working professionals are considered. It is also likely that there will be big differences between different groups of working professionals. These differences will need to be explored in future work.

The primary study is limited in the same way. Here, it would have been more desirable to have a more diverse sample; however, this would have conflicted with the need for a homogeneous sample for the meta-study. Since the meta-study was our main goal, we accepted the limitation of the primary study. That being said, there are early indications that computer science students can be acceptable proxies for professionals in developer studies [43, 49, 54, 107–112]. This sample was not selected for its representatives and thus should not be used to infer anything about non-students.

Since our study was performed in a laboratory environment with laboratory PCs, we have an unknown amount of bias in our results. This is particularly relevant to the meta-study. While the low amount of attempted security in the non-prompting condition seems plausible in light of the many password database compromises, we have no way of confirming that we are measuring the same effect. It is possible that the low amount of attempted security in the non- prompted group is not due to participants' lack of awareness that passwords should be hashed and salted, but rather to a lack of concern for passwords in a study environment. In fact, we received statements to this effect in the exit survey. Of the 20 non-prompted participants,

- two attempted to implement a secure solution but failed,
- two thought it was secure despite not having done anything to secure it themselves,
- two stated that they did not implement security because it was not part of the task,
- three stated that the functionality was more important to them than security,

- three were aware that security was needed but did not give any reason why they did not implement it, and
- eight were not aware that hashing and salting were important for password storage.

We must point out that the above statements are based on self-reporting by the participants. False reporting is possible in both directions. Participants who might not have been aware of the need for security might have felt embarrassed and stated that they did know but chose not to implement it and made up a reason for it. It is also possible that a participant who did know stated otherwise so as not to have to explain why security was not implemented. We must acknowledge these limitations.

We only conducted Bonferroni-Holm correction for our main hypotheses. For the rest of the exploratory analysis, we accepted the higher probability of type 1 errors to lower the risk of type 2 errors. Thus, new findings need to be confirmed by replication before they are used.

5.8. Summary

In this chapter, we presented an extension of our qualitative developer study on password storage [113] (Chapter 4). The extension had the dual goal of generating insights into the effect of design for developer studies, as well as furthering the understanding of why developers struggle to store passwords securely. We examined seven main hypotheses concerning both the primary study and the meta-study. We also compared our quantitative extension to the qualitative results of Chapter 4. Our results suggest that prompting or not prompting participants allows us to study different aspects of student developer behavior. Prompting can be used to discover usability problems of security APIs and test improvements with a straightforward study setup. Non-prompting (i.e., deception), though, might be used to research why developers do not add security without study countermeasures or being prompted. However, more work is needed to validate the ecological validity of deception in this context. We also found many participants use Google to answer survey questions. This is potentially very damaging to studies that do not account for this effect and one of many reasons we see for using qualitative research methods such as interviews to study developers.

The next step in this research endeavor is to design an experiment to study the effect of prompting on professionals. Since it is unrealistic to expect even a small number of working professionals to sacrifice a full day to take part in a lab study, a different study design was needed. Chapters 6 and 7 describe follow-up studies on password storage that were conducted with freelancers and developers employed in companies. These studies were conducted online. The security behaviors of all participants were compared to create a stronger foundation for usable security and privacy studies.

6. A Password-Storage Field Study with Freelance Developers

Disclaimer: *The contents of this chapter were published as part of the publication "'If you want, I can store the encrypted password.' A Password-Storage Field Study with Freelance Developers" (Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI) 2019) [117]. This paper was produced in cooperation with my co-authors Anastasia Danilova, Eva Gerlitz, Emanuel von Zezschwitz, and Matthew Smith. For the programming task, I used the same web application frame that I designed and implemented for the studies presented in Chapter 4 and Chapter 5. The results of those studies were published in [113] and in [116], respectively. The study design also was based on those studies, but it was adapted for this study by Matthew Smith, Eva Gerlitz and me. Emanuel von Zezschwitz conducted a pilot study which led to important changes in the study design. Eva Gerlitz, Ben Swierzy, and I prepared the website used in this study. This study was performed by Eva Gerlitz under my supervision. Anastasia Danilova and I analyzed participants' survey data. I conducted an analysis of participants' programming code, assisted by Eva Gerlitz. Statistical analysis was a joint work with Anastasia Danilova and Matthew Smith. I discussed key insights and implications of the work with Matthew Smith. Finally, I prepared the paper for publication in cooperation with Anastasia Danilova and Matthew Smith.*

6.1. Motivation

In recent years, many researchers have investigated end-user password creation, password use and knowledge [58, 60, 62–64, 66, 89–91], as well as password composition policies and their effects [56, 61, 75, 170]. Due to end-users' weak password habits (e.g., password reuse) [55, 209], attackers can focus on services with low security mechanisms to target services with high security standards [98]. 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, we conducted two studies in which 40 computer science students were asked to implement the registration process for a university social network (Chapters 4 and 5) [113, 116]. In a qualitative (Chapter 4) [113] and quantitative study (Chapter 5) [116], we 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-prompted 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-prompting conditions. It could also have led the participants to take security less seriously in the prompting 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 [94, 95, 102, 103] in which developers made mistakes just like the students did in the study.

To shed light on this problem, we repeated the studies presented in Chapters 4 and 5 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.

6.2. Methodology

As a starting point we adopted the methodology and study design frame from the study presented in Chapter 5 [116], 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 prompting/non-prompting 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 the previous student studies (Chapters 4 and 5) were mostly homogeneous and we hoped to gain more insights by focusing on JSF.

6.2.1. First Pilot Study

In our first pilot study we used exactly the same task as described in Chapters 4 and 5 (see Section 2.1.2). 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 [113, 116], 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, which students received in the studies described in Chapters 4 and 5, was framed as a university social networking website, he assumed the task to be an exercise for university credit.[1]

6.2.2. 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 Appendix C.4 and C.5.

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).

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

Second Pilot Study

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

6.2.3. 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 Appendix C.1). 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 Appendix C.2 and is an extended version of the one used in the study presented in Chapter 5. 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. Additionally, since freelancers could view which projects were offered to previous developers, our first contact text was formulated in a generic way concealing any details of the project. Thus, we made sure freelancers did not wonder why we were hiring developers to work on the same task.

6.2.4. 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 used in Chapters 4 and 5 (see Section 2.2.1). 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.

6.2.5. Quantitative Analysis

Due to the adjusted study design, we were able to test three of the seven main hypotheses presented in Chapter 2.3.1. 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:

H-P1 Prompting has an effect on the likelihood of participants attempting security.[2]

H-G1 Years of Java experience have an effect on the security scores.

H-G2 If participants state that they have previously stored passwords, it affects the likelihood that they store them securely.

We used the same statistical tests as used in Chapter 5 and which are described in Section 2.3.5. We did not use multivariate statistics since we considered our sample size too small [210]. We used Bonferroni-Holm corrections for all tests concerning the same dependent variable. To ease

[2]While we were able to track security attempts in a lab setting, in a field study this information was not accessible. We did, though, consider the subset *secure = 1* for our analysis.

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.

6.2.6. Qualitative Analysis

We used *inductive coding* [211] to analyze our qualitative data from the open questions in the survey as well as the chat 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) [199] was calculated. The agreement was 0.91. A value above 0.75 is considered a good level of coding agreement [200].

6.2.7. 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:

1. $\mathbf{P_{100}}$: **P**rompting-**100**Euro
2. $\mathbf{N_{100}}$: **N**on-Prompting-**100**Euro
3. $\mathbf{P_{200}}$: **P**rompting-**200**Euro
4. $\mathbf{N_{200}}$: **N**on-Prompting-**200**Euro

Then we 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.

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
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 6.1.: Demographics of the 43 participants

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 6.1.

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.

6.3. 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.

6.4. Results of Freelancer Password Study

Table 6.2 and Table 6.3 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.

6.4.1. 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 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 6.3). For the secure solutions, 4 came from non-prompted participants and 13 from prompted. Consequently, task design had a significant effect, with

[3]Table 6.2 considers participant $N2_{100}$, who suspected the task was part of a research study. This participant was not included in Table 6.3 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**	$\mathbf{2^{10}}$	**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	10000	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	10000	Static	✓	✗	✗
N1$_{100}$	**0**	**100**	**1 Day**	**4min**	**0/1**	**0/6**	**BCrypt**	**184**	$\mathbf{2^{12}}$	**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	10000	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	65536	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**	**64000**	**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**	$\mathbf{2^{10}}$	**SecureRandom**	✓	✗	✗

Table 6.2.: 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.

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

Experience. Acar et al. [49] found a correlation between programming experience, whereas in the student study (Chapter 5) we did not. We postulated that the effect might not be visible due to the smaller range of experience seen in the student sample. Thus, here 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 =

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

Table 6.3.: Number of Secure Solutions per Condition

0.52, $cor - p$ = 0.52, OR = 0, CI = [0, 8.91], family = 3).

Copy and Paste. In Chapter 5, we reported a significant positive effect of copy and paste. Due to the fact that here 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.

A summary of our statistical analysis results is available in Table 6.4.

6.4.2. 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 Appendix C.3.

In each phase developers weight on different aspects before deciding to store user passwords securely. The request phase outlines one important finding of this work: "If you want security, ask for it." The implementation phase describes on which conditions and misconceptions freelancers base their decisions of how to store user passwords security as well as which information sources and library features could motivate them to consider security. Finally, the reflection phase describes different developer types we defined according to participants' statements about security, security standards and their programming practices. Based on their types, developers make decisions of how to solve programming tasks.

6.4.3. 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

H	**Sub-sample**	**IV**	**DV**	**Test**	**OR**	**CI**	p**-value**	$cor - p$**-value**
Omnibus test	-	Payment, prompting	Secure	FET	-	-	0.02^*	-
H-P1#	-	Prompting	Secure	FET	6.55	[1.44, 37.04]	0.01^*	0.03^*
H-G1#	-	Java experience	Security score	Kruskal-Wallis	-	-	0.21	-
H-G2#	-	Stored passwords before	Secure	FET	0	[0, 8.91]	0.52	0.52
H-G4	-	Payment	Secure	FET	2.29	[0.56, 10.20]	0.22	0.44
E-A1	Prompted = 1	Payment	Task acceptance	FET	0.24	[0.058, 0.90]	0.02^*	-
E-A2	Prompted = 0	Payment	Task acceptance	FET	0.56	[0.07, 3.42]	0.69	-
E-A3	-	Security request	Security score	Wilcoxon rank sum	-	-	0.83	-
E-A4	Prompted = 0	Sample (students, freelancers)	Secure	FET	0	[0,1.89]	0.12	-

Table 6.4.: Summary of statistical analysis
IV: Independent variable, **DV**: Dependent variable, **OR**.: Odds ratio, **CI**: Confidence interval, **E-A**: Exploratory analysis
All tests were conducted on security values before participants received a *security request* except for E-A3.
H-P1, H-G2 and H-G4 are corrected with Bonferroni-Holm correction (cor-p value).
# = Hypothesis from Section 2.3.1
* = Significant tests

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. Though, even when we instructed our participants to store the passwords securely, they often did not use appropriate techniques to do so. Further, $N6_{100}$ believed that the client needs to be aware of security when hiring freelancers:

> *"All is based on client requirement. The requirement should state [that the] password should be stored securely."*

6.4.4. 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

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

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*" ($N6_{100}$). This misconception shows that participants confuse encoding functions with hash functions by reducing them to visual representation, a phenomenon also shared by end users.

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 freelancers used symmetric encryption in order to store user passwords and some were very convinced that this was a good solution:

> "*[I] used 32-bit encryption with three-layer security on top of it. [...] 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., $N5_{100}$, $N11_{100}$, $P2_{200}$, $N8_{100}$, $N10_{100}$, $P3_{200}$). A phenomenon, which was also observed in the student sample (Chapters 4 and 5) and shows that freelancers do not update their knowledge as well. For instance, $N11_{100}$ declared,

> "*Passwords are stored after encryption, usually MD5. This makes it secure and almost impossible to compromise.*"

Additionally, $N11_{100}$ seemed to lack knowledge of attackers, as his comment demonstrates:

> "*Password is in database so users won't be able to access it.*"

$N10_{100}$ recommended to use MD5 as well:

> "*I use always some one way encryption algorithm to save user's password in database like md5. So that it can never be decrypted.[...] I recommend to use database methods like md5 for encrypting password.*"

Functionality First, Security Costs Extra

Similar to end users and participants from the student sample (Chapter 4) [113], 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 6.2.7).

Information Sources

Further, information sources were mentioned as reasons to choose and use an API to store passwords securely. $P4_{100}$ told us that Spring offers a good documentation and easy API, therefore he included Spring in his solution. Further, code examples and tutorials were mentioned as reasons to choose an API ($N4_{100}$) as well as the amount of documentation ($N6_{200}$, $N8_{200}$).

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 ($N1_{100}$, $N10_{100}$, $N3_{200}$, $N6_{200}$, $P4_{200}$). Similar to $P5_{100}$ and $P11_{200}$, $N10_{200}$ said:

> "*The classes and methods were available easily for usage with proper documentation [...].*"

6.4.5. 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 Appendix C.3 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.

Realist

P10_{200} was prompted for security and used `Base64` as his technique of choice to store user passwords securely. According to his solution, he acknowledged "*there is no optimal code in the world.*" A belief, which was shared by N6_{100}:

> "*There is always chances of improvement in code.*"

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 would store passwords securely even if not explicitly instructed to. For example, $N10_{200}$ argued:

> "*Passwords are secure fields, they should not be revealed to anyone and anywhere.*"

Additionally, $P5_{200}$ said:

> *"Yes I would have stored it securely even if it is not asked. Because I am aware user passwords are sensitive information."*

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

6.5. 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 recruited convenience samples such as computer science students [112, 113, 188, 193] or GitHub developers [48, 49, 172, 212, 213].

The response rate for GitHub users is rather low [48–50, 112, 172, 212]. For instance, of the 50 000 invited developers in [112], 302 took part in the study. In [48] 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 [49], 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 [50], 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.

6.5.1. Freelance vs. Student Sample

Requirements

Similar to the students from the study presented in Chapter 5 [113] (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 (Chapter 4) [113], shows that freelancers do not update their knowledge as well. Yet as we observed in Chapter 5 [116], 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.

6.5.2. 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 presented in Chapters 4 and 5 [113, 116], 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 [139] reported freelancers to be generally unreliable, Yamashita and Moonen [191, 192] 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. Further, some of the freelancers asked very detailed technical questions, that would be hard to answer for an employer without technical background.

6.5.3. Lessons Learned

We asked participants to store user data in our database as we hoped to gain insights about their workflow around functionality and security. Ten participants used their local database for code testing. Eight of them did this because the table was automatically created on the database when the program was run the first time, and they saw no reason to do the additional work of creating the table themselves. One participant stored the data on his local database due to language barriers, and another was unable to connect to our database. Of the remaining participants, at least 10 switched back and forth between their local and our database. Therefore, we found that providing a remote database represented additional overhead without delivering reliable results with regard to security observations.

6.6. Key Findings

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 Chapters 4 and 5 [113, 116], 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 (Chapter 4) [113] 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.

6.7. 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,[5] none of these participants implemented a secure solution.

6.8. Summary

In this chapter we present the results of a field study with 43 freelance developers recruited from Freelancer.com. Broadly speaking, we found similar results as we did in the lab study (Chapters 4 and 5) [113, 116]. We confirmed that task framing has a large effect. Importantly, we shed light on the statement made by many of the students in the previous 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.

This study provide some early indications for the ecological validity of security developer studies conducted with CS students in a lab. In order to provide deeper insights into this research field, the following chapter describes a password-storage study conducted with professional software developers working in different companies. The results suggest that freelancers' and company developers' behavior in regard to secure password storage can differ.

[5]But unluckily for security in general.

7. Examining a Password-Storage Study with Students, Freelancers, and Company Developers

Disclaimer: *The contents of this chapter were published as part of the publication "On Conducting Security Developer Studies with CS Students: Examining a Password-Storage Study with CS Students, Freelancers, and Company Developers" (ACM CHI Conference on Human Factors in Computing Systems (CHI) 2020) [118]. This paper was produced in cooperation with my co-authors Anastasia Danilova, Eva Gerlitz, and Matthew Smith. For the programming task, I used the same web application frame that I designed and implemented for the studies presented in the previous chapters. The study design also was based on those studies, but it was adapted for this study by me. For study design changes I was advised by Matthew Smith. I led the execution of the study and analyzed participants' programming code. Johanna Deuter, Manfred Paul, and Martin Welsch assisted me with the programming code analysis. Conducting the statistical analyses was a joint work with Anastasia Danilova and Matthew Smith. I conducted the qualitative analysis. Eva Gerlitz assisted with issues during the analysis of the study data. Finally, I prepared the paper for publication in cooperation with Anastasia Danilova and Matthew Smith.*

7.1. Motivation

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 [2]. In their research agenda, Acar et al. [105] argued that *ecological validity* issues are a major concern of usable security studies with developers. Similar to end-user goals [1, 4, 130, 131], security is considered as a secondary task by software developers [112, 113, 116, 117]. They are rarely security experts [2], and therefore task design in security developer studies can have a large influence on developers' programming practices [113, 116, 117]. Furthermore, the recruitment of professional software developers for research studies is challenging due to lack of time, spread out geographical locations, and high cost [43, 49, 54, 105, 112, 214, 215]. Consequently, previous conducted security software engineering research recruited computer science (CS) students, who are often studied out of convenience [43, 54, 112, 113, 116, 188, 193]. While there is evidence

	CS Students [113, 116]	**Freelancers [117]**	**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 [50] 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 [50]
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 7.1.: Comparison of password-storage studies conducted with CS students [113, 116], freelancers [117], and company developers.

that students behave similarly to professionals in software engineering studies [107–111], only limited knowledge exists on whether this holds true for security developer studies. In 2019, we [117] 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 presented in Chapter 6 [117] with employed professional software developers. We henceforth refer to these people as "company developers." With the freelancer study being a follow-up study of the student study (see Chapters 4 and 5) [113, 116], we present in Table 7.1 an overview of the previous studies and this study.

In 2017 and 2018, we asked computer science students to program the registration functionality for a university social platform in a laboratory setting (Chapters 4 and 5) [113, 116]. 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 [114], which offers supporting libraries for secure password storage, while the other half used JavaServer Faces (JSF) [115] 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, we replicated the study in 2019 with freelance developers recruited online on Freelancer.com [117]. The

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

framework variable was not investigated since all participants were asked to use the JSF framework. Within the pilot studies, we recognized that the university context of the study led participants to think they were solving a homework task for university credits. Therefore, we changed the study context. This time, we 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. We 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 chapter, we extended our work by conducting an online study with 36 regularly employed professional software developers. Like in the freelancer study, 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) [215], we refrained from the university context and opted for the start-up scenario. We adopted the study design of the freelancer study [117] 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 in the student study, we examined the variable framework with the new sample of professional developers.

With this work, we contribute on a primary level by providing insights into the password-storage practices of company developers. In addition, we discuss methodological implications for security developer studies on a meta-level by comparing password-storage practices across different developer samples: students, freelancers, and company developers. As suggested by Sjoberg et al. [215], we compared the results of students, freelancers, and professionals in *absolute* and *relative* terms. We found that company developers overall performed better than students with regard to security measures. However, 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).

7.2. 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 (Chapter 6) [117]; that study replicated yet a lab study with computer science students (Chapter 5) [116]. Table 7.1 shows a comparison of the previous studies and this study with regard to the study setup. In the students study, we asked participants to complete the registration functionality of a university social networking platform. We investigated whether prompting to security and using frameworks with different levels of secure password-storage support affected participants' solutions. After finishing the task, participants needed to fill out a survey covering their password security background knowledge, general security and IT skills, and their demographics. We found a significant effect for prompting, but not for the frameworks. Though, students claimed they would behave differently in a company. Therefore, we replicated the study with freelancers recruited on Freelancer.com without announcing the study purpose of the project. After recognizing in pilot studies, that the previous university context lead freelancers to believe they were hired to solve university homework, we changed the study context. This time, we posed as a start-up searching for a developer to complete the registration functionality for its sports photo sharing social networking platform. To make it more realistic, we provided participants a web presence of the social networking platform *SportSnapShare.com* (see Appendix C.5). After finishing the programming task, freelancers were informed about the study purpose and were invited to fill out a survey.

The recruitment of professional developers posed new challenges not faced in the study of freelancers, but it presented new opportunities as well. Some adjustments were made to the study frame for this specialized sample. Following the suggestions of Sjoberg et al. [215], one of our researchers, who works part-time in an IT company, recommended the project to her company. The company has 250 employees and provides services in diverse fields, including project management, software development, quality management, and IT security. To preserve confidentiality, this company is here referred to as "ITXcompany." ITXcompany was consulted to ensure that our study design decisions were reasonable. Developers at this company are deployed as externals in other companies throughout Germany. Therefore, it was not possible to assign the project to employees as a regular task in a laboratory setting or during their working time. Thus, we opted for the same online study design used in the freelancers study. 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. After the study was approved by ITXcompany's management board, the study was announced to software developers with Java experience.

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 7.1.

7.2.1. 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 with freelancers, we used the same task as that in our role-playing scenario, but informed participants about the fact that they were solving it within the scope of a research study (see Appendix D.7). Participants were also provided with the start-up's web presence (see Appendix C.5).

Study Compensation

In the original study (Chapter 5), students needed about 6-8 hours to finish the project. Thus, in the study with freelancers, we announced the project as a freelance job that would take 8 hours (Chapter 6). Based on freelancers' varied compensation proposals, we explored the effect of two payment levels: Participants were paid either 100 or 200 euros for the project (see Table 7.1). 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. We 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, 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 (Chapters 5 and 6), 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, we found that participants received a higher security score when they used Spring rather than JSF. We concluded that this was because Spring offers a security library

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

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 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 Chapters 4 and 5, we asked company developers to use either Spring or JSF to solve the task (see IV in Table 7.1).

7.2.2. Final Study

As displayed in Table 7.1, we explored two *independent variables (IV)*:

1. **IV-Framework:** Half the participants used Spring with supporting security libraries for password storage, whereas the other half used JSF without supporting libraries.
2. **IV-Prompting:** Half the participants were prompted to store user password securely, whereas the other half were told the study purpose is API usability investigation.

While, in the previous studies the IV *framework* was investigated with students, the IV *prompting* was explored with student and freelancer participants. We assumed that the results with students would differ from those with professionals in absolute terms. 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:

1. **P**rompting-**J**SF (PJ)
2. **N**on-Prompting-**J**SF (NJ)
3. **P**rompting-**S**pring (PS)
4. **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 (see Section 2.2.1) used in Chapter 6. As displayed in Table 7.1 (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),

[3]Previously called *security* (Chapters 4, 5, and 6). In order to avoid confusion between *secure* and *security*, the name of this variable security was changed to *score*.

- the iteration count for key stretching (0-1 point(s)),
- whether and how the salt was generated (0-3 points).

The detailed security scale can be found in Section 2.2.1. The occurrence of new practices to store passwords securely in the freelancer study lead us to extend the scale by several methods: both `Base64` and `symmetric encryption` solutions received 0 points for security, and `HMAC` was treated as a hash function with a non-random salt.

In Chapters 5 and 6, we showed that security requirements in the task description are crucial. Otherwise, most of the student and freelance developers did not consider secure password storage in their code. Therefore, we conducted a deeper analysis on security requests as fallows.

Security Request

As shown in Table 7.1, freelancers who submitted plain text solutions were requested to revise their submissions and to store user passwords securely (*SecRequest-P[laintext]*):

- **SecRequest-P:** "*I saw that the password is stored in clear text. Could you also store it securely?*"

Based on the start-up scenario in the freelancer study, we acted as a clueless employer, who only knows that user passwords should not be stored in plain text. Therefore, participants with `Base64` or `symmetric encryption` solutions did not receive the security request. If submitting plain text solutions, though, they received the security request. 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. They were rather formulated in a general way: "*We are specifically interested in the security aspects of password storage with Spring/JSF*" (Appendix A.5) and "*Please ensure that the user password is stored securely*" (Appendix A.6, A.7, and C.4). Acar et al. [112] demonstrated, however, that information sources can affect developers' security related programming practices. Particularly, compared to the informal documentation Stack Overflow [128], participants using official API documentation produced less functional but more secure code and vice versa. By contrast, in Chapter 5 we showed that participants who copied and pasted code from internet sources, such as Stack Overflow, all achieved secure solutions.

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

[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 [97, 153, 157, 216]. 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.

(*SecRequest-G[uideline]*) asking them to follow the recommendations of the National Institute of Standards and Technology (NIST) [97] or the Open Web Application Security Project (OWASP) [216]:

- **SecRequest-G:** "*I noticed, that you did not follow industry best practices, e.g., NIST (National Institute of Standards and Technology) or OWASP (Open Web Application Security Project), to securely store the end-user password. Could you please revise your submission and ensure that you follow industry best practices? You can find some information on OWASP on this website: [link1]*[5] *and information on NIST on this website: [link2]*[6] *in Section 5.1.1.2.*"

As first information source, we proposed to use the official "Digital Identity Guidelines on Authentication and Lifecycle Management" of NIST [97]. As second information source, we proposed the "Password Storage Cheat Sheet" of OWASP [216], which is a popular online community producing open source articles, methodologies, documentation, etc. in the field of web application security. We expected NIST to be a secure but hard to use documentation, whereas we assumed OWASP to be a secure but rather usable documentation. Reasons for this classification are the following. In comparison to the NIST source, the OWASP source is rather a short summary of the most important aspects for secure password storage. It provides a guidance on how and why user passwords should be stored securely and makes suggestions which hash algorithms should be used:

- "**Argon2** [157] is the winner of the password hashing competition and should **be considered as your first choice** for new applications;
- **PBKDF2** [149] when FIPS certification or enterprise support on many platforms is required;
- **Scrypt** [151] where resisting any/all hardware accelerated attacks is necessary but support isn't.
- **Bcrypt** [217] where PBKDF2 or Scrypt support is not available."

Additionally, the OWASP website provides usage proposal (code for copy and paste) for the implementation of `Argon2` in Java and PHP. By contrast, the "Digital Identity Guidelines" of NIST include a large document with a lot of information on the global topic of "Authentication and Lifecycle Management." Since we did not expect participants to read the whole NIST document and also wanted the effort of reading the document to be similar to OWASP, we mentioned where the relevant section on password storage security can be found: 5. Authenticator and Verifier Requirements/5.1.1.2 Memorized Secret Verifiers [97]. NIST gives examples of suitable key derivation functions, such as `PBKDF2` [149] or `Balloon` [218] and states "*A memory-hard function SHOULD be used because it increases the cost of an attack.*" Participants could use the information source of their choice. Figure 7.1 shows the security request procedure.

[5]`https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Password_Storage_Cheat_Sheet.md` [216]

[6]`https://pages.nist.gov/800-63-3/sp800-63b.html` [97]

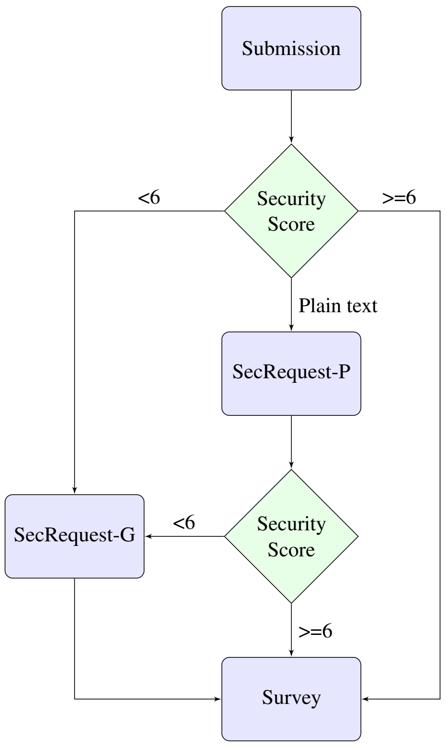


Figure 7.1.: Security request procedure
Participants received the security request SecRequest-P, if they submitted a plain text solution and the security request SecRequest-G if the solution scored less than 6 points. Participants with plain text solutions thus, could receive both requests.

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 (see Appendix B.2 and C.2). 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 detailed survey can be found in the Appendix D.6.

7.2.3. 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 [43, 49, 53, 105, 117, 214], 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

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 7.2.: Demographics of the 36 participants

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 [219], a German career-oriented social networking site similar to the American platform LinkedIn [220]; 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 7.2. 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).[7] 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: 11.8). Twelve participants reported that their organization has a security focus, of whom 6 also explicitly indicated to work in a team

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

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 Appendix D.1.

7.2.4. Evaluation

Code Analysis

To ensure data reliability [221, 222], 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 [223] or SecureRandom [224] 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 Section 2.3.1. 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:

- H-P1 - Priming has an effect on the likelihood of participants attempting security.
- H-F1 - Framework has an effect on the security score of participants attempting security.
- H-G1 - Years of Java experience have an effect on the security scores.
- H-G2 - If participants state that they have previously stored passwords, it affects the likelihood that they store them securely.

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. For the data analysis, we used the same tests as those utilized in Chapters 5 and 6 (see Section 2.3.5). 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 7.2.2). 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 Chapters 5 and 6 and this replication

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

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) [225]. All factors for our regression analysis are summarized in Table 7.3. For all regressions, we selected as final the model with the lowest AIC.

Qualitative Analysis

We analyzed the qualitative data from the open-ended questions in the survey through *inductive coding* [211]. 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) [199] was calculated. The agreement was 0.83. A value above 0.75 suggests a high level of coding agreement [200].

7.3. 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 the ITXcompany that its general performance would be kept confidential. Since we received only four valid submissions from the ITXcompany, we cannot make significant statements about the company's performance. 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

	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 7.4.: 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

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 (see Chapers 4 and 6). 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).

7.4. Results

Table 7.4 shows an overview of participants' submissions. An overview of participants' submission results per variable can be found in the Appendix D.2. 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 7.4b). 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 7.4c). 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).

A detailed evaluation of the submissions we received from the participants as their initial or successive submission after SecRequest-P is available in Table 7.5. Within participants' initial or

Participant	**Prompting**	**Framework**	**Working Time**	**Include Security**	**Active Working Time**	**SecRequest**	**Function**	**Length in bits**	**Iteration**	**Salt**	**Secure**	**Security Score**
PJ1	1	JSF	10 Days		5h		BCrypt	184	2^{10}	SecureRandom	1	6
PJ2	1	JSF	18h		4h		BCrypt	184	2^{10}	SecureRandom	1	6
PJ3	1	JSF	4 Days		7h	G	Blowfish				0	0
PJ4	1	JSF	22 Days		9h	G	PBKDF2 (SHA-1)	256	32	SecureRandom	1	5
PJ5	1	JSF	23.5 Days		20h	G	SHA3-512	512	1	SecureRandom	1	5
PJ6	1	JSF	3.5 Days		6h		PBKDF2 (SHA-512)	512	65 536	SecureRandom	1	6
PJ7	1	JSF	6 Days		9h	G	SHA-512	512	1	SecureRandom	1	5
PJ8	1	JSF	12 Days		20h	G	PBKDF2 (SHA-1)	128	65 536	SecureRandom	1	5
PJ9	1	JSF	6.5 Days		5h	G	PBKDF2 (SHA-1)	128	65 536	SecureRandom	1	5
PJ10	1	JSF	9 Days		6h	G	PBKDF2 (SHA-1)	128	65 536	SecureRandom	1	5
NJ1	0	JSF	26 Days		16h	G	PBKDF2 (SHA-1)	128	65 536	Static	1	4
NJ2	**0**	**JSF**	**3 Days**	**7h**	**10h**	**P + G**	**sym. Encryption (DES)**				**0/0**	**0/0**
NJ3	**0**	**JSF**	**1 Day**	**3h**	**7h**	**P + G**	**MD5**	**128**	**1**		**0/1**	**0/1**
NJ4	0	JSF	8.5 Days		6h	G	SHA-512	512	1	Static	1	4
NJ5	**0**	**JSF**	**15 Days**	**13h**	**3h**	**P + G**	**MD5**	**128**	**1**		**0/1**	**0/1**
NJ6	**0**	**JSF**	**16h**	**2.5 Days**	**2h**	**P + G**					**0/0**	**0/0**
NJ7	**0**	**JSF**	**8 Days**	**11h**	**10h**	**P + G**	**PBKDF2 (SHA-1)**	**128**	4 096	**SecureRandom**	**0/1**	**0/4.5**
NJ8	**0**	**JSF**	**1.5 Days**	**12h**	**6h**	**P + G**	**SHA-256**	**256**	**1**	**SecureRandom**	**0/1**	**0/5**
PS1	1	Spring	14.5 Days		5h		BCrypt	184	2^{10}	SecureRandom	1	6
PS2	1	Spring	5 Days		2h		BCrypt	184	2^{10}	SecureRandom	1	6
PS3	1	Spring	9.5 Days		18h	G	SHA-512	512	5 000	Static	1	4.5
PS4	1	Spring	3.5 Days		6h		BCrypt	184	2^{10}	SecureRandom	1	6
PS5	1	Spring	2.5 Days		6h		BCrypt	184	2^{10}	SecureRandom	1	6
PS6	1	Spring	4.5 Days		5h		BCrypt	184	2^{10}	SecureRandom	1	6
PS7	1	Spring	9 Days		3h		BCrypt	184	2^{16}	SecureRandom	1	6
PS8	1	Spring	10.5 Days		6h		BCrypt	184	2^{10}	SecureRandom	1	6
PS9	1	Spring	8 Days		4h		BCrypt	184	2^{10}	SecureRandom	1	6
PS10	1	Spring	1.5 Days		7h		BCrypt	184	2^{10}	SecureRandom	1	6
PS11	1	Spring	9 Days		5h		BCrypt	184	2^{10}	SecureRandom	1	6
NS1	**0**	**Spring**	**2 Days**	**8h**	**4h**	**P**	**BCrypt**	**184**	2^{10}	**SecureRandom**	**0/1**	**0/6**
NS2	**0**	**Spring**	**5 Days**	**1 Day**	**4h**	**P**	**BCrypt**	**184**	2^{10}	**SecureRandom**	**0/1**	**0/6**
NS3	**0**	**Spring**	**8 Days**	**9 Days**	**3h**	**P**	**BCrypt**	**184**	2^{10}	**SecureRandom**	**0/1**	**0/6**
NS4	**0**	**Spring**	**9.5 Days**	**6h**	**5h**	**P + G**	**MD5**	**128**	**1**	**Username**	**0/1**	**0/3**
NS5	0	Spring	2.5 Days		4h		BCrypt	184	2^{10}	SecureRandom	1	6
NS6	0	Spring	8 Days		12h		BCrypt	184	2^{10}	SecureRandom	1	6
NS7	**0**	**Spring**	**16.5 Days**	**19h**	**3h**	**P**	**BCrypt**	**184**	2^{10}	**SecureRandom**	**0/1**	**0/6**

Table 7.5.: Evaluation of participants' submissions
Bold: Participants who at first delivered an insecure solution and received the security request SecRequest-P.
Working time participants took to submit their first solution.
Include Security: Time participants needed to add security after SecRequest-P.
Active Working Time: Self-reported active working time reported by participants for their final submissions.

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. Most of our participants used hashing (and salting) as methods to store user passwords securely, whereas two participants used symmetric encryption approaches and thus, received 0 points for security. 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

Participant	Prompting	Framework	Include Security	Function	Length in bits	Iteration	Salt	Secure	Security Score	NIST	OWASP
PJ3	1	JSF	4 Days	Argon2	256	25	SecureRandom	1	7		✓
PJ4	1	JSF	5.5 Days	Argon2	256	40	SecureRandom	1	7	✓	
PJ5	1	JSF	8 Days	Argon2	256	40	SecureRandom	1	7		✓
PJ7	1	JSF	6 Days	SHA-512 + Argon2	256	40	SecureRandom	1	7		
PJ8	1	JSF	3.5 Days	PBKDF2 (SHA-1)	192	1 000	SecureRandom	1	5.5		✓
PJ9	1	JSF	14h	PBKDF2 (SHA-1)	128	34 721	SecureRandom	1	5	✓	
PJ10	1	JSF	14.5 Days	Argon2	128	40	SecureRandom	1	7	✓	✓
NJ1	0	JSF	7 Days	Argon2	256	2	SecureRandom	1	7		
NJ2	0	JSF	1 Day	BCrypt	184	2^{16}	SecureRandom	1	6	✓	
NJ3	0	JSF	5h	Argon2	256	10	SecureRandom	1	7		✓
NJ4	0	JSF	2 Days	BCrypt	184	2^{10}	SecureRandom	1	6	✓	✓
NJ5	0	JSF	1.5 Days	SHA-3	512	1		1	2		
NJ6	0	JSF	2 Days	Argon2	256	40	SecureRandom	1	7		✓
NJ7	0	JSF	1.5 Days	Argon2	256	1 000	SecureRandom	1	7		✓
NJ8	0	JSF	2 Days	Argon2	256	5	SecureRandom	1	7	✓	✓
PS3	1	Spring	9.5 Days	Argon2	256	40	SecureRandom	1	7		✓
NS4	0	Spring	1h	Argon2	256	40	SecureRandom	1	7		✓

Table 7.6.: Evaluation of participants' submissions after SecRequest-G
NIST/OWASP: Participants indicated to have used NIST/OWASP for their submissions.

and received 6 points for security.

An overview of participants' solutions after receiving SecRequest-G (<6 points) is available in Table 7.6. 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. The detailed parameter choices of `Argon2` can be found in Table 7.7. A number of participants 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.

In the next sections we present further analysis on the findings and our hypotheses results. A summery of our statistical analysis is available in Table 7.8.

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 7.4a 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

Participant	**Function**	**Hash Length (bits)**	**Iteration**	**Salt**	**Salt Length (bits)**	**Salt Generation**	**Memory**	**Threads**
PJ3	Argon2	256	25	1	128	SecureRandom	128 000	4
PJ4	Argon2	256	40	1	128	SecureRandom	128 000	4
PJ5	Argon2	256	40	1	128	SecureRandom	128 000	4
PJ7	SHA-512 + Argon2	256	1 + 40	1	128	SecureRandom	128 000	4
PJ10	Argon2	256	40	1	128	SecureRandom	128 000	4
NJ1	Argon2	256	2	1	128	SecureRandom	16 384	2
NJ3	Argon2	256	40	1	128	SecureRandom	128 000	4
NJ6	Argon2	256	40	1	128	SecureRandom	128 000	4
NJ7	Argon2	256	1000	1	128	SecureRandom	128000	2
NJ8	Argon2	256	5	1	128	SecureRandom	65 536	1
PS3	Argon2	256	40	1	128	SecureRandom	128 000	4
NS4	Argon2	256	40	1	128	SecureRandom	128 000	4

Table 7.7.: Participants' Argon2 parameter choices

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).

Java Experience (H-G1)

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).

Password-storage Experience (H-G2)

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.

H	Sub-sample	IV	DV	Test	O.R.	CI	p-value	$cor - p$-value
H-P1#	-	Prompting	Secure	FET	46.33	[4.74, 2434.13]	<0.001*	<0.001*
H-G1#	-	Java experience	Score	Kruskal-Wallis	-	-	0.69	0.69
H-G2#	-	Stored Passwords Before	Secure	FET	0.41	[0.04, 2.69]	0.44	0.44
H-F1#	secure = 1	Framework	Score	Wilcoxon rank sum	-	-	0.003*	0.006*
E-A1	-	Team Security Focus	Secure	FET	1.31	[0.17, 16.06]	1	-
E-A2	-	Company Security Focus	Secure	FET	0.34	[0.06, 1.83]	0.16	-
E-A3	-	Company Security Size	Secure	FET	1	[0.19, 4.99]	1	-
E-A4	-	Framework	API Usability [50]	Wilcoxon rank sum	-	-	0.08	-
E-A5	-	Score	API Usability [50]	Pearson Cor.	-	[-0.42, 0.23]	0.54	-
E-A6	-	Solve Security Tasks Regularly	Secure	FET	0.60	[0.10, 3.03]	0.72	-
E-A7	secure = 1	Java experience	Score	Kruskal-Wallis	-	-	0.64	-

Table 7.8.: Summary of statistical analysis
IV: Independent variable, **DV**: Dependent variable, **O.R.**.: Odds ratio, **CI**: Confidence interval, **E-A**: Exploratory analysis
All tests were conducted on security values *before participants received any security requests*.
H-P1 and H-G2 as well as H-G1 and H-F1 are corrected with Bonferroni-Holm correction (cor-p value).
# = Hypothesis from Section 2.3.1
* = Significant tests

Framework Usability

Similar to the study presented in Chapter 5, we calculated the API usability scores as suggested by Acar et al. [50] 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. [53], 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]).

	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 7.9.: Number of secure solutions per condition

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 Appendix D.3.

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, 8 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.

7.5. Meta-level Analysis Across Samples

For our meta-level analysis, we compared the password-storage results of students (see Chapter 5), freelancers (see Chapter 6) 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

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 7.10.
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 7.11.
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.

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 7.9 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 7.10 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$\times$ higher odds that the participants' solution would include secure storage of user passwords. In comparing samples, student participants were only 0.08$\times$ 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 7.2 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

IV	DV	St.Test	Students [116]	Freelancers [117]	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 7.12.: 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).

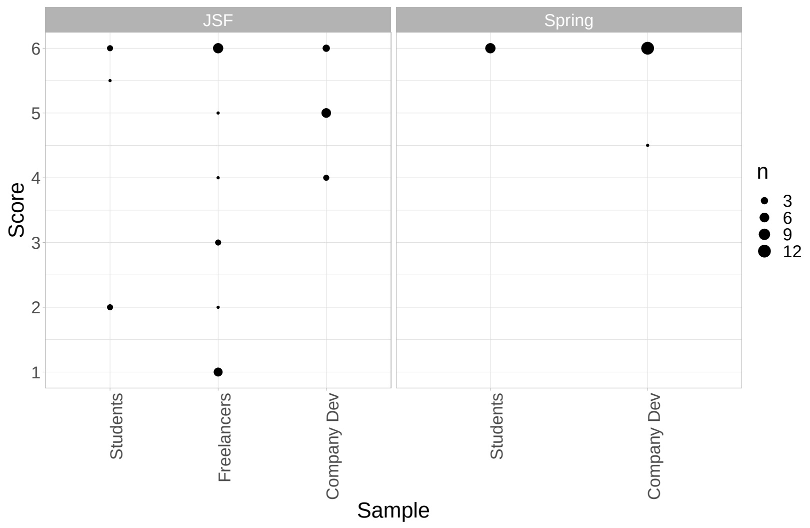


Figure 7.2.: Scores across samples (secure = 1)
Company dev = company developer. The size of the points symbolizes how many participants from a sample gained the corresponding scores considering initial and secure solutions only. In the freelancer sample only the JSF framework was tested.

conditions, although they showed a greater variety of scores for JSF compared to company developers and students. Table 7.11 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.

Table 7.12 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 7.2 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. Based on the restricted time in the lab, students might also submit more functional solutions, if have been given more time. This could suggest that research on APIs can offer similar insights by recruiting both students and company developers for usability studies.

7.6. Discussion

Ecological validity issues are a major concern of usable security studies with developers [105]. While there is evidence that students behave similar to professionals in software engineering studies [107–111], limited knowledge exist whether this holds true for security developer studies. Our 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. [215], 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.

7.7. 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 with students (see Chapter 5). 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.

7.8. Summary

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 our password-storage study presented in Chapters 4, 5, and 6 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.

With this work not only new insights for security practices with regard to password storage of company developers were given, but also methodological insights for security developer studies were discussed. The next chapter provides a bigger picture of the conducted work and its implication for future research. Further, take-aways, suggestions and a concept will be presented in order to improve the security of software.

8. Discussion and Recommendations

This chapter summarizes the findings presented in this thesis. It also discusses the primary results on password-storage security and the meta-level methodological implications of these studies. Based on this discussion, recommendations for future research will be provided.

This thesis has presented four password-storage studies conducted with CS students, freelancers, and company developers. When comparing these studies, it is important to remember that the student studies were conducted in a lab, while the freelancer and company developer studies were conducted online. Furthermore, the students and company developers were informed that they were participating in a research project at the beginning of the study, while the freelancers were hired for a programming job and only informed that it was a research project after they had submitted the tasks.

First of all, both students and freelancers were less likely than company employees to initially submit "secure"[1] solutions in which user passwords were at least hashed. Table 8.1 summarizes the distribution of the different sample submissions (without considering the variables framework and prompting). In the study with students, 30% of the participants submitted secure solutions and 70% submitted non-secure solutions. In contrast, 67% of developers employed at companies submitted secure solutions and 33% submitted non-secure solutions. Chapter 7 demonstrates that company developers performed significantly better than students in regard to secure password storage. However, all the sample sizes are small, so these results should not be given undue weight. Still, these findings provide some early indications of the security behavior of different developer samples regarding password storage. Table 8.2 summarizes the security of participants' code in relation to the variable of security prompting. This table demonstrates that the numbers of secure solutions shown in Table 8.1 depend heavily on whether participants were asked to store user passwords securely or not. Most participants in all samples who were not prompted did not consider the task to be strongly related to security. The findings of the freelancer study demonstrate that this result was not a study artifact since the freelancers were hired for a programming task and believed they were working for a start-up, not participating in a study. Nevertheless, they behaved similarly to students in terms of security practices (see Chapter 7).

The security practices implemented to store user passwords differed among the samples. A number of students and freelancers stored user passwords in plaintext or chose poor parameters.

[1]It should be mentioned once more that the term "secure" is used very loosely in this thesis. Here, "secure" simply means that participants used the appropriate practice to store hashed passwords (without considering the algorithm choice and salting). The degree of security implemented in different solutions is reflected in the security score (see Section 2.2).

CS Students n = 40		**Freelancers n = 42**		**Company Developers n = 36**	
Non-secure	Secure	Non-secure	Secure	Non-secure	Secure
70%	30%	60%	40%	33%	67%

Table 8.1.: Percentage of (non-)secure solutions considering initial submissions

	CS Students			**Freelancers**			**Company Developers**		
	n	Non-secure	Secure	n	Non-secure	Secure	n	Non-secure	Secure
Non-Prompting	20	100%	0%	21	81%	19%	15	73%	27%
Prompting	20	40%	60%	21	38%	62%	21	5%	95%

Table 8.2.: Percentage of (non-)secure solutions for the variable prompting considering initial submissions

Some freelancers also used inappropriate mechanisms such as `symmetric encryption` or `Base64`. While some company developers also chose poor practices, they still performed better than students or freelancers in absolute terms. Still, all sample groups struggled to achieve industry-standard security[2]. The interview and survey analysis showed that the students and freelancers lack knowledge of password storage security. Both of these samples had misconceptions about password storage security and often relied on outdated information when choosing algorithms. However, using an API with opt-in security made a difference. Freelancers used only JSF to solve the task, so it was not possible to draw conclusions for this sample. However, students and company developers working with Spring achieved higher security scores than participants who used JSF (see Chapter 7, Table 7.12).

These results suggest several conclusions and recommendations with regard to the primary-level analysis:

- **If you want security, ask for it.** Even when carrying out a task in which security is clearly critical, such as user password storage, it seems that software developers often fail to think about and implement security on their own. Thus, it is not safe to assume that developers will implement appropriate security measures even when carrying out security-related tasks. Most non-prompted participants in these studies did not store user passwords securely. Even if participants obviously knew that passwords should not be stored in plaintext in the database, indicated by their interview, survey, and programming code comments, they often skipped the additional effort of including security code. These results indicate that developers perceive security as overhead and a secondary task. Consequently, if an assigned task does not include instructions about security, they may decide to skip the additional work. Furthermore, even if helpful tutorials or information sources are available, developers often do not take the time to research aspects of security unless specifically requested to do so.

[2]See Section 2.2.1.

- **Secure password storage is a complex task that requires crypto knowledge.** Even when participants were instructed to securely store user passwords, they often chose inappropriate techniques or weak security standards. This highlights the most important finding of this thesis: that securely storing user passwords is indeed a non-trivial task, and developers often struggle with it. This finding offers an explanation for the frequent security breaches observed in real life [94, 95, 102, 103]. It is not realistic to expect developers to stay constantly up-to-date on the latest security trends. There are many security use cases, and developers must often work on various tasks. They are rarely security experts, so they should not be expected to always be aware of current security practices for every use case.

- **Providing information sources such as OWASP or NIST can help improve software security.** Chapter 7 demonstrates that if company developers are provided with information sources such as OWASP or NIST, they can implement state-of-the-art security measures for user password storage. However, an employer without a technological background cannot be expected to know which tasks require special attention to security or which information sources their developers should use.

- **"Technology should adapt to its users rather than requiring users to adapt to technology" [2].** The findings summarized above underline Green and Smiths' [2] statement that "technology should adapt to its users" and not the other way around. The present studies with students and company developers demonstrate that APIs that offer support for secure password-storage can take the burden off users and help them produce secure software. However, if these functionalities are not included by default and developers are not aware of them, they tend to not use them. Therefore, it is extremely important to support developers in security tasks and to provide usable security APIs with secure defaults. Most of the participants' submissions were as secure as the suggestions in the tutorials, the API/framework security defaults, or the information sources participants used to solve the task. Therefore, researchers should focus on improving the usability and security of these tools. To do this, however, it is necessary to investigate developers' security practices in order to identify which tools require the most attention.

- **Providing software developers with a website or assistance tool that offers examples of secure code that they can copy and paste might improve software security.** In addition to usable APIs with opt-out security features and secure defaults, it would be desirable to provide developers with a general tutorial from which they could extract ready-to-use programming code for web applications. In the study described in Chapter 5, participants copied and pasted solutions from various online sources. Although most participants in that study achieved secure solutions by copying and pasting programming code, recent studies have shown that such behavior can also promote security vulnerabilities [112, 186]. Often, developers lack the time and knowledge to verify which sources provide high security standards and which should be avoided. Therefore, a website or tool provided by a trustworthy authority that includes state-of-the-art, ready-to-use security code might improve software security by taking the burden off developers. With regard to authentication, this website or tool could ask developers which features they are searching for. For instance, code examples might cover secure password storage, password storage policies, and two-factor authentication (2FA) [97]. Developers could then choose which standards they need to confirm, such as NIST, OWASP, or the requirements of the German "Bundesamt für Sicherheit in der Informationstechnik (BSI)" [226]. *CogniCrypt* is an example of such a tool; it was proposed by Krüger et al. [227, 228]. CogniCrypt can support developers with cryptographic tasks by generating security-critical programming code and conducting static analysis in the background. This seems to be a promising approach, but usability research with software developers is needed.

This thesis also has some methodological implications and provides some recommendations for developer studies on usable security:

- **Researchers might consider prompting participants for security when investigating the security features of APIs.** The most important finding of this thesis with regard to the meta-level analysis is that security prompting makes a difference (see Table 8.2). If researchers are testing the usability of a security API but do not instruct participants to think about security, participants might not use the security features.

- **Deception might increase the ecological validity of security studies with developers.** Further research is needed on the use of deception in security studies with developers. The findings of the studies described in this thesis suggest that deception simulates a more realistic environment, as employers without a background in technology or security rarely ask for secure implementation but rather for software functionality.

- **Researchers might consider including functionality requirements in security tasks to increase the ecological validity of security studies with developers.** The length of the tasks used in developer studies needs more attention from researchers. As discussed in Chapter 5, including aspects of functionality and security in a programming task results in longer tasks. Especially if professionals are recruited for a study, a longer programming task (such as the task used in this thesis) can result in a lower sample size. However, including functionality requirements might create a more realistic scenario and thus probably more ecologically valid findings.

- **Researchers might choose a qualitative approach rather than a quantitative one if sample sizes are expected to be small.** The findings of the qualitative study with students described in Chapter 4 provided already good indications for the findings of the follow-up quantitative study (Chapter 5). This is particularly relevant to the study described in Chapter 7, in which company developers performed significantly better than students in absolute terms and received significantly higher security scores than students and freelancers. Therefore, qualitative studies investigating the absolute performance of APIs might offer valuable insights without the need to recruit a large number of professionals. This is particularly promising in light of the statement above that the use of longer programming tasks that include functionality and security might improve the ecological validity of security studies with developers. With a qualitative design, fewer professionals would be required to work on longer programming tasks.

- **Researchers might recruit students when they want to investigate security behavior in relative terms.** Chapter 7 demonstrated that the findings regarding relative behavior applied to all developer samples used in all studies. This means that students, freelancers, and company developers submitted significantly less secure solutions if they were not prompted and that students and company developers (freelancers were not tested) achieved higher security scores with Spring than with JSF. Since the usable security and privacy community is concerned with increasing secure development rather than with which samples perform better, these are promising results for the ecological validity of developer studies conducted with students.

- **Researchers need to consider which developer group is targeted by their research.** It is very important that researchers strategically plan their studies and consider ecological validity. For example, if, in real life, a given task would be done by freelancers hired online and not by developers employed by a company, a research sample of students holds promise for investigating absolute as well as relative performance (see Chapter 6).

However, these insights are drawn from one security use case, and more research is needed to validate them.

Overall, this research makes several contributions to the usable security and privacy community. This study has examined how different groups of developers (students, freelancers, and company employees) perceive the task of user password storage, which is critical to security. It has also examined how these groups of developers perform with regard to state-of-the-art security standards and which approaches might help improve software security. Developers are the end users of tools, frameworks, APIs, and tutorials employed in development tasks. However, the findings of this thesis suggest that developers are rarely aware that their programming code needs to address software security as well as functionality. Furthermore, even if they are aware of this, they perceive security as overhead and a secondary task. Therefore, more research is needed to determine whether lessons learned from end-user research might be adapted to software developers as well. In fact, the findings of this thesis support the call of previous researchers [2, 20, 105] for future studies investigating the security-related behavior of administrators, software developers, security designers, and cryptographers. It is essential to examine how these parties perceive security, because they are responsible for security layer by layer. Many security issues experienced by end users might be preventable if the actors involved in software creation were included in research on usable security and privacy. However, it is still unclear how such studies should be conducted to ensure ecological validity. Administrators, software developers, and cryptographers might have different participant requirements than typical end users. This thesis has presented several methodological approaches to such research, providing some early indications of the ecological validity of different approaches to security studies with developers.

9. Conclusion

This thesis contributes to existing knowledge on usable security research by examining the specialized user group of software developers. Authentication is an important issue for the usable security and privacy community, and research with end users has been conducted for decades. While end users' security mistakes often affect only their own sensitive data, programming security mistakes made by software developers can threaten millions of end users' data. Therefore, it is essential to understand software developers' motivations, attitudes, and perceptions, as well as the practices on which they base security decisions. User password storage is one of the tasks most commonly performed by software developers, but password databases are nevertheless prone to security vulnerabilities. Therefore, this thesis has examined developers' security behavior with regard to user password storage. Two studies, one qualitative and one quantitative, were conducted with CS students recruited from the University of Bonn. Both of these studies were conducted in a lab. Additionally, studies with freelancers and company developers were conducted online. Freelancers were hired online via Freelancer.com, while company developers were contacted through their companies and through the German business social networking platform Xing.

Participants were instructed to complete programming code for the registration function of a social networking platform. Half the participants were instructed to store user passwords securely, while the other half were not. In addition, half the participants used a framework that provided opt-in support for secure password storage (Spring), while the other half had to implement secure password storage on their own using the framework JSF. In the student study, none of the participants stored user passwords securely without prompting. Furthermore, students often employed poor practices for password storage, especially when they had to choose hash algorithms and parameters for secure password storage on their own rather than using the crypto libraries of the framework with opt-in security. Copy-and-paste programming codes from online sources (such as tutorials) had a positive effect on secure solutions. However, some students reported that they would have behaved differently with regard to security implementation if they were solving the task for a real company rather than in a study. Nevertheless, the follow-up study with freelancers showed that the results of the student study were not just a study artifact. The freelancers' programming submissions indicated similar behavior to that of students in the lab with regard to password storage security. Freelancers had misconceptions about password storage security and thus often chose weak or inappropriate security practices. Additionally, they tended to completely ignore security if they were not prompted. The final study with company developers further suggested that, in relative terms, the security behavior of all three samples was similar. For example, students, freelancers, and company developers were all more likely to consider security if prompted, and all tested samples achieved better security results when using an API that provided support for password storage security (freelancers were not tested with different APIs). In absolute terms, however, company developers performed better than students and freelancers with regard to password storage security practices. However, none of the participants achieved state-of-the-art password storage security unless they were provided information sources with ready-to-use programming examples.

The findings of this thesis suggest that developers struggle to securely store user passwords. However, developers who used APIs with security support for password storage or who accessed information sources with secure, ready-to-use code tended to achieve better software security. Still, developers who were not instructed to consider password storage security often seemed to perceive adding security to their programming code as an undesirable overhead which they tended to omit. This is understandable since secure user password storage is far more complex than is generally believed. Without cryptographic knowledge and an understanding of this specific task, it seems that developers are often at a loss to choose appropriate methods, hash functions, and parameters. The study findings have shown that developers struggle with these decisions, and the large amount of supposed helpful information on the Internet seems to be overwhelming. Developers have to deal with far more complex tasks than the average end user, and they are usually not security experts [2], so it takes more time and significant effort for them to address security issues. Additionally, developers are the end users of APIs, so they are prone to make security mistakes if APIs lack usability or appropriate security defaults. Therefore, in an extension of the request, "Don't Blame the User" [229], I would like to make a suggestion:

"**Don't Blame Developers, Either!**"

This thesis considers secure user password storage and APIs in Java by examining the behavior of samples of student, freelance, and company developers. Future work should consider different security use cases, programming languages, APIs, and developer groups with different characteristics in order to verify the current results and to provide more insights into the motivations, attitudes, and practices affecting the security behavior of software developers. It would be especially interesting to explore whether assistance tools and APIs with opt-out security would improve software security by taking the burden of security-related decisions off developers. The findings in this thesis have also shown that company developers are more likely than freelancers and students to submit secure solutions with better parameter choices. This raises an additional question: Why did company developers generally store passwords more securely than students and freelancers in absolute terms? While company context might be the key, further research is needed to examine different hypotheses about this finding. It might be necessary to study the mental models of actors involved in software development and software creation with respect to perceptions, decisions, and practices around security. Such a study might use a similar approach to that used by Krombholz et al. with administrators [20].

While this thesis offers valuable insights about ecological validity of security studies with developers, more research is needed to verify that these results also apply to different security tasks. In particular, further studies are needed to investigate whether qualitative instead of quantitative studies are sufficient for specific use cases. The findings of the student study described in this thesis provide an early indication that qualitative studies might reveal essential insights, possibly eliminating the need for quantitative studies in some cases. Furthermore, further research should be conducted on the relationship of deception to the ecological validity of studies of developer security behavior. This is especially interesting since it cannot be assumed that developers' employers have the necessary IT or security knowledge to advise developers on software security. It might therefore be more realistic to design security studies so that developers are not aware of the true purpose of the study, such as security. Still, ethical issues need to considered in this context. Task framing is another aspect that requires researchers' attention. Short, security-focused tasks might not reveal all issues. Longer programming tasks that include aspects of functionality and security might be needed to achieve valid results. Finally, more research is needed on absolute and relative security findings. Researchers must rethink study designs considering which concept adequately reflects their approach. The absolute and relative findings on security behavior described in this thesis need to be replicated before tangible recommendations can be made.

A. Appendix

A.1. Recruitment Invitation

Participant invitation for scientific studies
Dear computer science students, We are conducting several scientific studies, in which you can participate and earn 100 euros!
Details:
The institute for Computer Science of the University of Bonn is looking for motivated Computer Science Students (over 18), who want to take part in a scientific study with the topic of Web Development in Java and to earn some extra money. The goal of the study is to test the usability of different Java web development APIs. In the study you will be asked to implement parts of a web application. Basic knowledge of Java and the IDE Eclipse is required. The study will last 8 hours. Afterwards we will conduct a short interview and ask some questions about the tasks. The interview will be audio recorded to facilitate the evaluation of results. The aim of the study is not to test your knowledge but the usability of the APIs. All data gathered during the study will be anonymized. Anonymized data and quotes may be published as part of a scientific publication. Your consent will be requested on the day you participate. You will be payed 100 euros for taking part in the study. Knowledge required: IDE Eclipse, JAVA Interested? Please fill in the following questionnaire to register your interest: *LINK* We are looking for a good mix of skills, so please fill out the questionnaire as accurately as possible. We are looking for up to 120 participants. Registering your interest does not guarantee participation.

A.2. Pre-Screening Questionnaire

We used a seven-point rating scale according to [194].

1. Gender: *Female/Male/Other/Prefer not to say*
2. Which university are you at? *University of Bonn/Other: [free text]*
3. In which program are you currently enrolled? *Bachelor Computer Science/Master Computer Science/Other: [free text]*
4. Your semester: [free text]

5. How familiar are you with Java?
   1-Not familiar at all - 7-Very familiar
6. How familiar are you with PostgreSQL?
   1-Not familiar at all - 7-Very familiar
7. How familiar are you with Hibernate?
   1-Not familiar at all - 7-Very familiar
8. How familiar are you with Eclipse IDE?
   1-Not familiar at all - 7-Very familiar

A.3. Survey for Students

- *Expectation* asked before solving the task:
  What is your expectation? Overall, this task is?
  1: Very difficult - 7: Very easy
- *Experience* asked after solving the task:
  Overall, this task was ...?
  1: Very difficult - 7: Very easy
- *Security Expertise*: see Section A.3.1
- How often have you stored passwords in the software you have developed (Prompted Group: apart from this study)?
  1: Never - 7: Every time
- How would you rate your background/knowledge with regard to secure password storage in a database?
  1: Not knowledgeable at all - 7: Very knowledgeable
- Do you think that you stored the end-user passwords securely?
- The Web framework (Spring/JSF) supported me in storing the end-user password securely. **1: Strongly disagree - 7: Strongly agree**
- The Web framework (Spring/JSF) prevented me in storing the end-user password securely. **1: Strongly disagree - 7: Strongly agree**

A.3.1. Security Expertise

Q1 I have a good understanding of security concepts.
1: Strongly disagree - 7: Strongly agree

Q2 How often do you ask for help facing security problems?
1: Never - 7: Every time

Q3 How often are you asked for help when somebody is facing security problems?
1: Never - 7: Every time

Q4 How often do you need to add security to the software you develop in general (Prompted Group: apart from this study)?
1: Never - 7: Every time

A.4. Semi-structured Interview

- Do you think that you have stored the end-user passwords securely
  - No
    - Why?
    - *Non-Prompted Group:* Were you aware that the task needed a secure solution?
    - What would you do, if you needed to store the end-user password securely?
  - Yes (If yes, further questions below)
- What is the purpose of hashing?
- What is the purpose of salting?
- Did you hash the end-user password?
- Did you salt the end-user password?
- Did the Web framework *(Spring/JSF)* support you in storing the end-user password securely? Please explain your answer.
- Did the Web framework *(Spring/JSF)* prevent you from storing the end-user password securely? Please explain your answer.
- Imagine you were working in a company and you had exactly the same task as you got today with exactly the same task description and time constraint. Would you have solved the task in exactly the same way as you solved it today?
- Did you solve the task functionally?
- Did you solve the task securely?

Further Interview Questions:

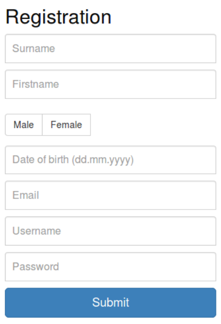
(If participants believe they have stored the end-user password securely.)

- *Non-Prompted Group:* How did you become aware of the necessity of security in this task? At which point did you decide to store the end-user password securely?
- What did you do to store the user password securely?
- Please name all steps taken in order to store the end-user password securely
- Do you think your solution is optimal? Why or why not?
- What were the general problems you encountered when implementing secure end-user password storage?
- *Prompted Group:* Do you think you would have stored the end-user passwords securely if you had not been told about it? Please explain your answer.

A.5. Study Task (with Security Prompting)

Welcome! *We would like to invite you to participate in our study and thus make a valuable contribution to our research. Our research goal is to study the usability of the Web Framework **Spring/Java Server Faces (JSF)**.*
(We are specifically interested in the security aspects of password storage with Spring/JSF.)

Task:
Imagine, the University of Bonn wants to offer a **social network**, which enables students and members of the university to communicate and interact with each other (similar to Facebook). For this purpose, a platform has to be provided allow users to **register** to the social network.
The university has employed a team of developers who have already implemented parts of the platform. For instance, a front-end designer has created this interface:



A database administrator provides a database access using **PostgreSQL** (Object-Relational Database Management System (ORDBMS)) and **Hibernate** (Object-Relational Mapping (ORM) Framework). Another team member started to implement the program logic, but left the team recently.
Please help the team and complete the partially available implementation. In order to do so, please implement the registration process of the application.
(We are specifically interested in the security aspects of password storage with Spring/JSF.)

*In order to solve the task, you are allowed to use **any kind of source available on the Internet that may be helpful or has valuable information**. Consequently, you can search and read every website, which you believe is helpful. Further implementation hints can be found on the next page.*

*Please fill in the following survey **before** working on the task: LINK_TO_SURVEY*

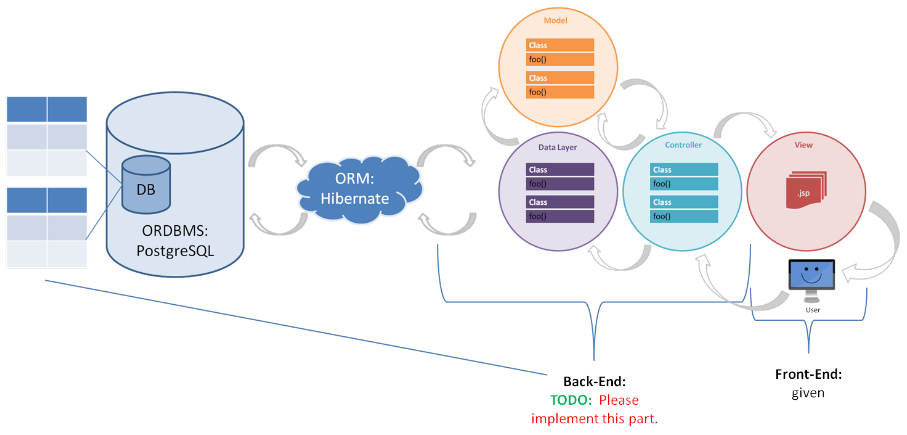
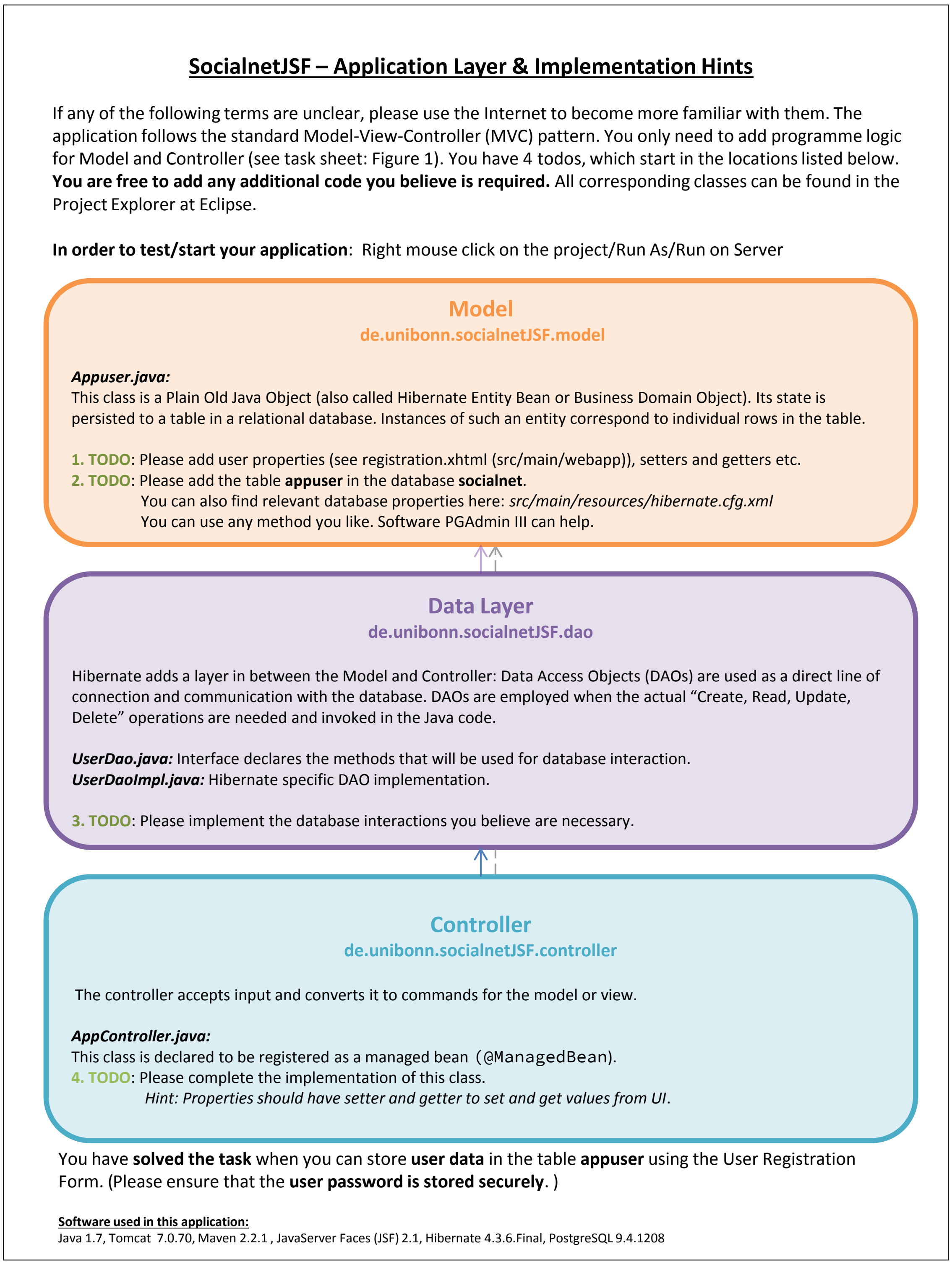


Figure 1: Back-End and Front-End

A.6. Implementation Hints for JSF (with Security Prompting)



SocialnetJSF – Application Layer & Implementation Hints

If any of the following terms are unclear, please use the Internet to become more familiar with them. The application follows the standard Model-View-Controller (MVC) pattern. You only need to add programme logic for Model and Controller (see task sheet: Figure 1). You have 4 todos, which start in the locations listed below. **You are free to add any additional code you believe is required.** All corresponding classes can be found in the Project Explorer at Eclipse.

In order to test/start your application: Right mouse click on the project/Run As/Run on Server

Model
de.unibonn.socialnetJSF.model

Appuser.java:
This class is a Plain Old Java Object (also called Hibernate Entity Bean or Business Domain Object). Its state is persisted to a table in a relational database. Instances of such an entity correspond to individual rows in the table.

1. TODO: Please add user properties (see registration.xhtml (src/main/webapp)), setters and getters etc.
2. TODO: Please add the table **appuser** in the database **socialnet**.
You can also find relevant database properties here: *src/main/resources/hibernate.cfg.xml*
You can use any method you like. Software PGAdmin III can help.

Data Layer
de.unibonn.socialnetJSF.dao

Hibernate adds a layer in between the Model and Controller: Data Access Objects (DAOs) are used as a direct line of connection and communication with the database. DAOs are employed when the actual "Create, Read, Update, Delete" operations are needed and invoked in the Java code.

UserDao.java: Interface declares the methods that will be used for database interaction.
UserDaoImpl.java: Hibernate specific DAO implementation.

3. TODO: Please implement the database interactions you believe are necessary.

Controller
de.unibonn.socialnetJSF.controller

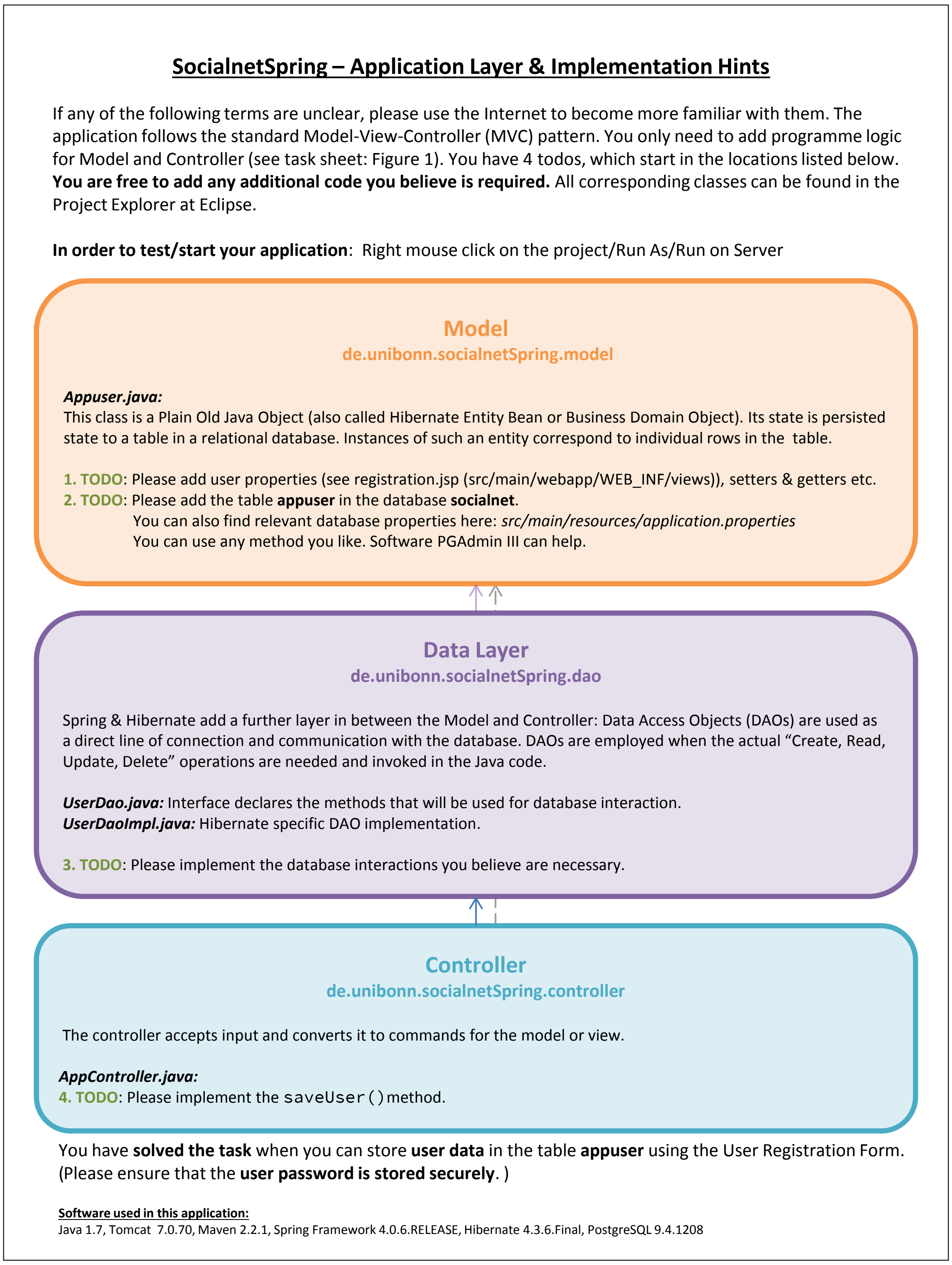
The controller accepts input and converts it to commands for the model or view.

AppController.java:
This class is declared to be registered as a managed bean (`@ManagedBean`).
4. TODO: Please complete the implementation of this class.
Hint: Properties should have setter and getter to set and get values from UI.

You have **solved the task** when you can store **user data** in the table **appuser** using the User Registration Form. (Please ensure that the **user password is stored securely**.)

Software used in this application:
Java 1.7, Tomcat 7.0.70, Maven 2.2.1 , JavaServer Faces (JSF) 2.1, Hibernate 4.3.6.Final, PostgreSQL 9.4.1208

A.7. Implementation Hints for Spring (with Security Prompting)



SocialnetSpring – Application Layer & Implementation Hints

If any of the following terms are unclear, please use the Internet to become more familiar with them. The application follows the standard Model-View-Controller (MVC) pattern. You only need to add programme logic for Model and Controller (see task sheet: Figure 1). You have 4 todos, which start in the locations listed below. **You are free to add any additional code you believe is required.** All corresponding classes can be found in the Project Explorer at Eclipse.

In order to test/start your application: Right mouse click on the project/Run As/Run on Server

Model
de.unibonn.socialnetSpring.model

Appuser.java:
This class is a Plain Old Java Object (also called Hibernate Entity Bean or Business Domain Object). Its state is persisted state to a table in a relational database. Instances of such an entity correspond to individual rows in the table.

1. TODO: Please add user properties (see registration.jsp (src/main/webapp/WEB_INF/views)), setters & getters etc.
2. TODO: Please add the table **appuser** in the database **socialnet**.
You can also find relevant database properties here: *src/main/resources/application.properties*
You can use any method you like. Software PGAdmin III can help.

Data Layer
de.unibonn.socialnetSpring.dao

Spring & Hibernate add a further layer in between the Model and Controller: Data Access Objects (DAOs) are used as a direct line of connection and communication with the database. DAOs are employed when the actual "Create, Read, Update, Delete" operations are needed and invoked in the Java code.

UserDao.java: Interface declares the methods that will be used for database interaction.
UserDaoImpl.java: Hibernate specific DAO implementation.

3. TODO: Please implement the database interactions you believe are necessary.

Controller
de.unibonn.socialnetSpring.controller

The controller accepts input and converts it to commands for the model or view.

AppController.java:
4. TODO: Please implement the `saveUser()` method.

You have **solved the task** when you can store **user data** in the table **appuser** using the User Registration Form. (Please ensure that the **user password is stored securely**.)

Software used in this application:
Java 1.7, Tomcat 7.0.70, Maven 2.2.1, Spring Framework 4.0.6.RELEASE, Hibernate 4.3.6.Final, PostgreSQL 9.4.1208

B. Appendix

B.1. Entry Survey for Students

Before solving the task, participants were asked questions Q5 - Q8 from the pre-screening questionnaire (Appendix A.2) one more time for consistency reasons. Additionally, they were asked two further questions:

1. *Expectation*:
   What is your expectation? Overall, this task is
   1-Very difficult - 7-Very easy
2. How familiar are you with JavaServer Faces (JSF)/Spring?
   1-Not familiar at all - 7-Very familiar

B.2. Exit Survey for Students

Questions asked after solving the task:

1. *Experience*
   Overall, this task was
   1-Very difficult - 7-Very easy
2. Do you think your solution is optimal? *No / Yes*
   – Why do you think your solution is (not) optimal? [free text]
3. I have a good understanding of security concepts.
   1-Strongly disagree - 7-Strongly agree
4. How often do you ask for help facing security problems?
   1-Never - 7-Every time
5. How often are you asked for help when somebody is facing security problems?
   1-Never - 7-Every time
6. How often do you need to add security to the software you develop in general (Prompted group: apart from this study)?
   1-Never - 7-Every time
7. How often have you stored passwords in the software you have developed (Prompted group: apart from this study)?
8. How would you rate your background/knowledge with regard to secure password storage in a database?
   1-Not knowledgeable at all - 7-Very knowledgeable

9. Do you think that you stored the end-user passwords securely?
   No / Yes
   - If Yes:
     * What did you do to store the passwords securely? [free text]
     * Do you think your solution is optimal? *No / Yes*
       · Why do you think your solution is (not) optimal? [free text]
     * Prompted group: Do you think you would have stored end-user passwords securely, if you had not been told about it? Please explain your decision. [free text]
   - If No:
     * Why do you think that you did not store the passwords securely? [free text]
     * Non-Prompted group: Were you aware that the task needed a secure solution? *No / Yes*
     * What would you do, if you needed to store the end-user passwords securely? [free text]
10. Did you use libraries to store the end-user passwords securely? *No / Yes*
    - If Yes:
      * Which libraries did you use to store the end-user passwords securely (in this study)? [free text]
      * Please name the most relevant library you have used to store the end-user passwords securely (in this study). [free text]
      * You have identified {*participant's answer*} as the most relevant library to store end-user passwords securely. How would you rate its ease of use in terms of accomplishing your tasks functionally / securely? *1- Very Difficult - 7- Very Easy*
        Please explain your decision. [free text]
      * Usability scale for {*participant's answer*} (see Section B.2.1)
11. JSF/ Spring supported me in storing the end-user password securely. *1 - Strongly disagree - 7- Strongly agree*
    Please explain your decision. [free text]
12. JSF/ Spring prevented me in storing the end-user password securely. *1 - Strongly disagree - 7- Strongly agree*
    Please explain your decision. [free text]
13. JSF/ Spring: Usability scale (see B.2.1); the term *library* was replaced by *framework*.
14. Have you used Java APIs / libraries to store end-user passwords securely before? *No / Yes*
    - If Yes:
      * Which Java APIs / libraries to store end-user passwords securely have you used before? [free text]
      * What is your most-used API / library for secure password storage? [free text]
      * How would you rate its ease of use in terms of accomplishing your tasks functionally? *1- Very Difficult - 7- Very Easy*
        Please explain your decision. [free text]

* How would you rate its ease of use in terms of accomplishing your tasks securely? *1- Very Difficult - 7- Very Easy*
  Please explain your decision. [free text]

B.2.1. Usability Scale [50]

By contrast to [50] we dropped the option "does not apply" for the last two questions, Q10 and Q11. Used scale in our study:

Please rate your agreement to the following questions on a scale from ’strongly agree’ to ’strongly disagree.’ (Strongly agree; agree; neutral; disagree; strongly disagree). Calculate the 0-100 score as follows: 2.5 * (5-Q_1 + $\sum_{i=2..10}$ (Q_i-1)); for the score, Q11 is omitted.

– I had to understand how most of the assigned library works in order to complete the tasks.
– It would be easy and require only small changes to change parameters or configuration later without breaking my code.
– After doing these tasks, I think I have a good understanding of the assigned library overall.
– I only had to read a little of the documentation for the assigned library to understand the concepts that I needed for these task.
– The names of classes and methods in the assigned library corresponded well to the functions they provided.
– It was straightforward and easy to implement the given tasks using the assigned library.
– When I accessed the assigned library documentation, it was easy to find useful help.
– In the documentation, I found helpful explanations.
– In the documentation, I found helpful code examples.

Please rate your agreement to the following questions on a scale from ’strongly agree’ to ’strongly disagree’. (Strongly agree; agree; neutral; disagree; strongly disagree).

– When I made a mistake, I got a meaningful error message/exception.
– Using the information from the error message/ exception, it was easy to fix my mistake.

B.2.2. Demographics

– Please select your gender. *Female/Male/Other/Prefer not to say*
– Age: [free text]
– What is your current occupation? *Student Undergraduate/Student Graduate/Other: [free text]*
– At which university are you currently enrolled? *University of Bonn / University of Aachen*
– Which security lectures did you pass in your Bachelor/Master programme? *(To select)/Other: [free text]*
– Currently, do you have a part-time job in the field of Computer Science? If yes, please specify: [free text]

- How many years of experience do you have with Java development? *< 1 year/ 1 - 2 years/ 3 - 5 years/ 6 - 10 years/ 11+ year*
- What is your nationality? [free text]
- Thank you for answering the questions! If you have any comments or suggestions, please leave them here: [free text]

C. Appendix

C.1. Playbook

To keep the communication with freelancers consistent, we used a uniform question/answer catalog. *P* indicates the question of the participant and *R* indicates the answer of the researcher.

Project Offer

– 1st Message: *R: Hi xyz, I noticed your profile and would like to offer you my project. We can discuss any details over chat.*

– 2nd Message: *R: Hey, we are developing a social networking website to share pictures with family and friends. People need to register to this website in order to be able to share their pictures. Your task would be to program the registration functionality for this website in the back-end. The project is in Java and uses Hibernate and JSF, our database uses postgreSQL. The front-end and some parts of the program logic are already developed. What do you think? Kind regards, ...*

– Over project bidding: *R: Hi. We are happy that you want to work on our project. We would like to give you the code as a ZIP file. *sending files**

– When participant accepts project: *R: Happy to hear that!*

Deadline

– *P:* What is the final delivery? *R:* What do you think how long you need to solve the task? *P:* *** *R:* Ok, that's fine. Thank you

– Deadline exceeded: *R:* Hi, could you give us a status update?

Password-related Questions

– *P:* Should I implement security/secure password storage? *R:* Yes, please!

– Participant handed in insecure code: *R:* I saw that the password is stored in clear text. Could you also store it securely?

– *P:* Is *** fine? *R:* Whatever you would recommend/use!

– *P:* "I have so much experience on Java/Hibernate/what you mentioned..." *R:* Perfect, sounds good!

- *P*: Cannot find password encryption in the requirements, can you tell me where it is written? I might have missed it." *R:* It is not in there, that is true. But could you add it?

General Questions

- *P:* Is it homework? *R:* No, does it look like it is?
- *P:* Can I build it from scratch? *R:* You can solve the task as you prefer.
- *P:* Do I have to use Eclipse? *R:* You can use an IDE of your choice.
- *P:* If I have some questions for your project, can I ask to you? *R:* Sure!
- *P:* Do you use Skype/...? *R:* No, sorry!
- About the budget: *R:* Unfortunately our budget is only .. euro right now. If not enough: *R:* But I'll keep you in mind, if we cannot find somebody else. Sorry!
- *P:* Is it one time development. or are there any future jobs? *R:* For this project it would be one time development, but if you like working with us, we might ask you to help us in the future as well.
- *P:* What happened to your previous developer? *R:* Unfortunately for us, he got a very good job offer.
- P wants email for communication. *R:* If you have any questions, you can just leave them here and I will answer as soon as I see them, if that is ok for you?
- *P:* I have seen your living site with your url *R:* Yeah, we put that online (even though it is not finished yet), so you get to know us a little bit and know who we are.
- *P:* Do you have server where we can upload this code for you to test. *R:* None that I have access to. Would it be possible for you to send me a video or screen-shots so I can see that it is working on your computer?
- *P:* I have a suggestion. Can't I access to your site source directly? As you know, local environment is different from server environment. and if I work there, we can check it online. / etc. *R:* Unfortunately, I do not have access to the source code right now. Is it ok for you to work on your local environment?
- *P:* Do you want me to build Registration on your website or should I do it locally on my machine and hand over it to you? *R:* It would be great if you could do it locally on your machine and hand it over!
- *P:* May I check the code and get back to you tomorrow/later/... *R:* Yes, sure! Take your time.
- P did not work on our db: *R:* Could you also make it work on our database?
- *P:* Once user registers, do we need to send verification email also and once he clicks on that, we will make user status as Active. *R:* No, we only need the data to be stored in our database for now!
- *P:* Which version control service do you use? *R:* Not any yet, as we are still at the beginning of the project.
- *P:* I need to see the ER diagram *R:* We do not have that yet. Is it necessary? *P:* .. *R:* Could you please create a single table for now and I will talk to my colleagues about the rest?

- *P:* Do you require the login functionality as well? Should I implement it as a further task? What else except registration will be needed? *R:* No, thank you!/Please only program the registration functionality.
- *P*: Password is in database, so users wont be able to access it" *R:* And what if somebody get access to the database?

Implementation and Technical Questions

- *P:* Are you using a SSL/TLS server? *R:* Yes.
- *P:* In java there is servlet or jsp? *R:* Yes!
- *P:* There are errors in compile *R:* The errors occur, because the files are not complete, that's what we would need you to do
- *P:* How do I install the project? *R:* Our developer worked with Eclipse, it should be possible to import the files as existing maven project.
- *P:* Can I use additional APIs/frameworks? *R:* Yes, you can use any additional APIs or framework you like.
- *P:* What exactly do I need to do? *R:* All fields in the registration form should be stored in a database.
- *P:* Can you provide me db credentials? *R:* You can find all relevant information in the task sheet, which is included in the ZIP file.

Review for Participants

- *R:* Very good communication, delivered on time. It was nice working with him!

Survey Request

Hello,
thank you very much for completing the task. We have one further request for you. We are researchers from the University of Bonn. The task you worked on is part of a scientific study we are conducting to help developers write more secure code. For an additional 20 euro we would like to invite you to fill out a survey related to the task you worked on. It will take roughly 20 to 30 minutes. The data will be processed pseudonymously. After the survey is complete all data will be anonymized and and there will be no identifying information published in any form. If you wish we can inform you concerning the results of our research. In this case please supply us with an email address under which we can reach you. We would be grateful if you can complete the survey here: [Link] Your study id is: [...]
Thank you again!
With kind regards,...

C.2. Survey for Freelancers

We adapted the exit survey presented in Section B.2 to the freelance context and expanded it by the following questions:

– Did you create the code on your own or in a team? *[I worked on my own; I worked in a team.]*
– Can you please fill out this questionnaire together with all people who did the coding? *[Ok, my team is here; Sorry, I can't get my team right now, but I was involved in the coding process; Sorry, I can't get my team right now and I was not involved in the coding process.]*
– Which IDE did you use to solve the task?*[free text]*
– Why did you not implement your suggestions? *[free text]*
– At any point, did you think this task could be part of a scientific study? *[Yes, No]*
  * If yes: When did you think this task could be part of a scientific study? *[free text]*
  * If yes: Why did you think this task could be part of a scientific study? *[free text]*
– What is your current main occupation? *[Freelance developer, Industrial developer, Industrial researcher, Academic researcher, Undergraduate student, Graduate student, Other]*
– Do you have a university degree?*[Yes, No]*
– At which university/universities were/are you enrolled? *[free text]*
– What was/is your subject? *[free text]*
– Were/Are you taught about IT-security at university? *[Yes, No, I don't recall.]*
– Were/Are you taught about IT-security extramural?*[Yes, No]*
  * If yes: Where were/are you taught about IT-security extramural? *[free text]*
– What was your main source of learning about IT-security? *[free text]*
– How did you gain your IT skills? *[free text]*
– How did you gain your IT-security skills?*[free text]*
– What type(s) of software do you develop? *[Web applications, Mobile applications, Desktop applications, Embedded applications, Enterprise applications, Other (please specify)]*
– How many years of experience do you have with Java development? *[Integer]*
– How many years of experience do you have with software development in general? *[Integer]*
– What country do you live in? *[free text]*
– Are you in contact with other freelancers working on freelancer.com? *[Yes, No]*
  * If yes: How do you communicate with other freelancers working on freelancer.com? *[free text]*
– How do you rate the payment of the implementation task? *[Way too little, Too little, Just right, Too much, Way too much]*
– How do you rate the payment of the survey? *[Way too little, Too little, Just right, Too much, Way too much]*

- If you wish to be contacted by our research group to be informed about the study results, you can provide us your email address. The email address will be stored separately from the study data. *[free text]*
- If you wish to be contacted by our research group for further studies in the future, you can provide us your email address. The email address will be stored separately from the study data. *[free text]*

C.3. Qualitative Analysis

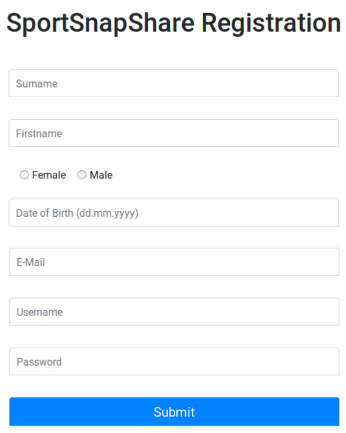
Information on our qualitative analysis of the open questions of the survey as well as the chat interactions during development is available in Table C.1.

C.4. Task (with Security Prompting)

Hello,

thank you so much for helping us finishing our project!

We are developing a social networking website to share pictures with family and friends. People need to register (email, name, gender, birth date, username and password) to this website in order to share their pictures. As we already wrote in the project description, we need help with finishing the registration logic for our social networking website. You can find our website here: Link



The front-end is already developed. Another developer started to implement the program logic (MVC), but left the team recently. **We need you to complete the partially available implementation by programming the registration functionality in the back-end.** ***(Please ensure that the user password is stored securely.)***

According to our former developer, the following things have to be done:

Setting up the environment:
The following software will be needed:
1. Tomcat server
2. Optional: Eclipse IDE for Java EE Developers
3. Optional: pgAdmin

Prepare the project & the database:
If you use Eclipse, it could look the following way:

1. Import the existing project as Existing Maven Project.
2. Add Maven Dependencies to the project in Eclipse: Right mouse click on the project/Properties/Deployment Assembly/Add/Java Build Path Entries → choose Maven Dependencies → Finish→ Apply

3. Connect to database (For example in pgAdmin: File/Add Server…)
You will find all necessary information (host, port, DB Name, username and password) in the *hibernate.cfg.xml* file (src/main/resources).

Implementing the registration process:
This was our developer's TODO list, it might not be complete, but hopefully it will help you understand what already was developed and what is left to be done. Figure 1 shows how the different parts are communicating with each other.

In order to test/start your application: Right mouse click on the project/Run As/Run on Server

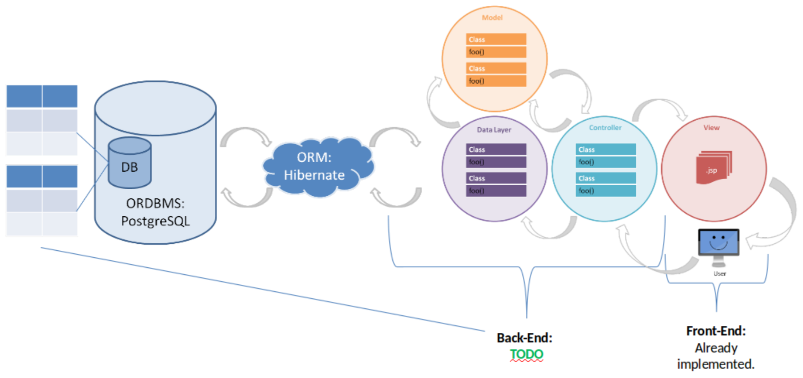


Figure 1

Util:
1. Done: Hibernate configuration
2. Done: Implement method: getSessionFactory()

View:
1. Done: Create registration.xhtml
2. Done: Create successRegistration.xhtml

Database:
1. TODO: Add table *appuser* in the database

Model – Appuser.java
1. TODO: Add user properties (see registration.xhtml)
2. TODO: Add getters & setters

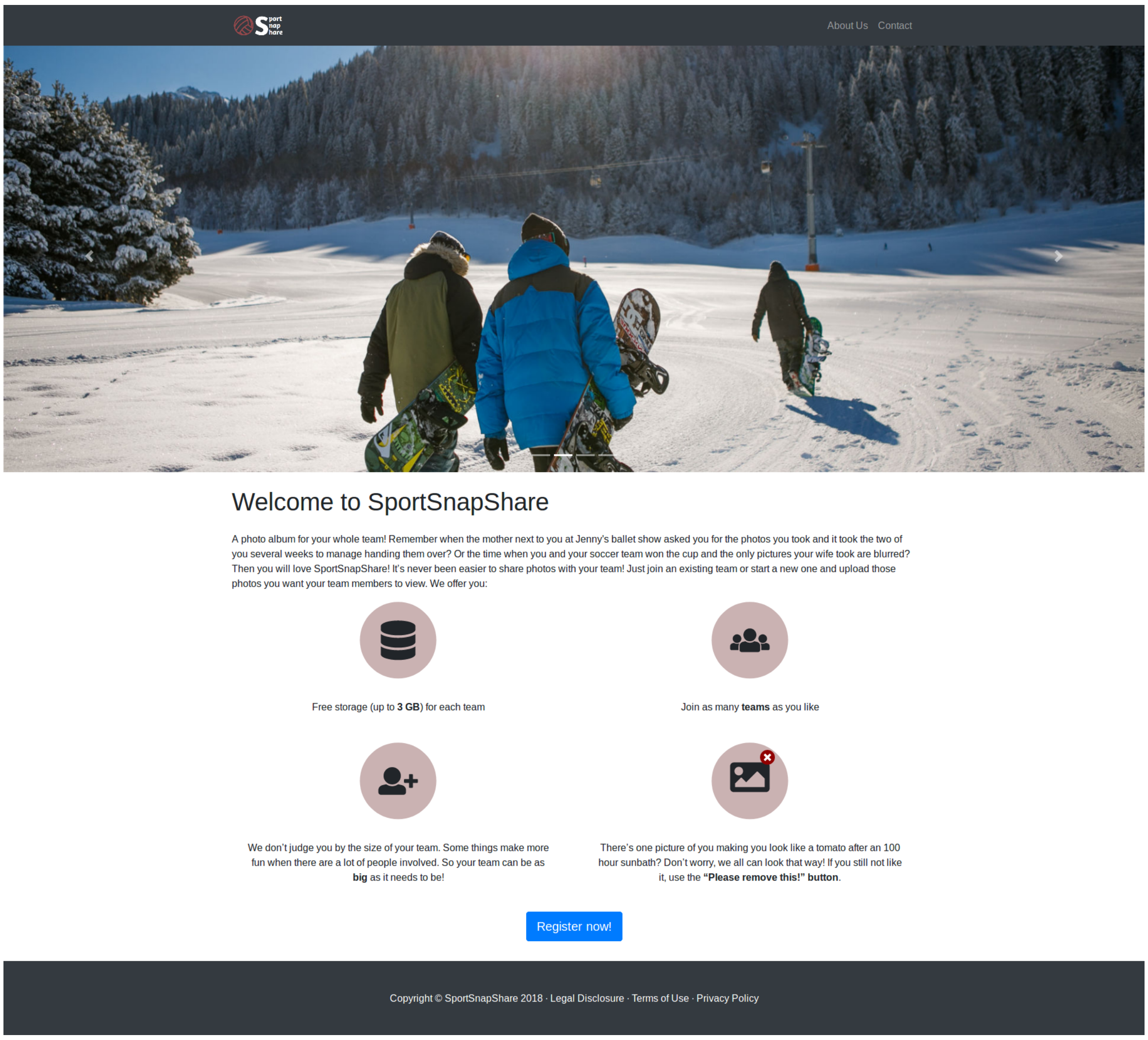
DAO – UserDAO.java
1. Done: Implement mehtod: validateUsername()
2. Done: Open session
3. TODO: Implement method: save()

Controller – AppController.java
1. TODO: Add variables for properties
2. TODO: implement method: saveUser()

(Please ensure that the ***user password is stored securely.)***

C.5. Web Presence

Figures C.1, C.2, C.3, and C.4 show snapshots of our start-up website.



Welcome to SportSnapShare

A photo album for your whole team! Remember when the mother next to you at Jenny's ballet show asked you for the photos you took and it took the two of you several weeks to manage handing them over? Or the time when you and your soccer team won the cup and the only pictures your wife took are blurred? Then you will love SportSnapShare! It's never been easier to share photos with your team! Just join an existing team or start a new one and upload those photos you want your team members to view. We offer you:

Free storage (up to **3 GB**) for each team

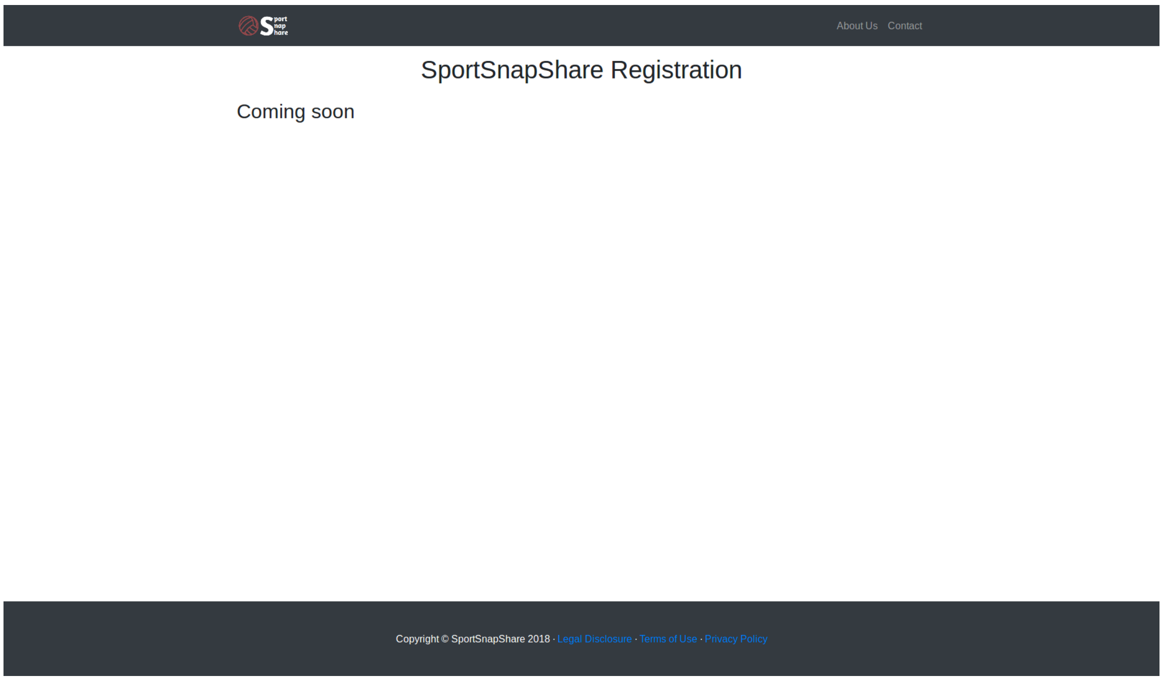
Join as many **teams** as you like

We don't judge you by the size of your team. Some things make more fun when there are a lot of people involved. So your team can be as **big** as it needs to be!

There's one picture of you making you look like a tomato after an 100 hour sunbath? Don't worry, we all can look that way! If you still not like it, use the **"Please remove this!" button**.

Copyright © SportSnapShare 2018 · Legal Disclosure · Terms of Use · Privacy Policy

Figure C.1.: Index Page
Sport Photo: ©Visit Almaty [230]

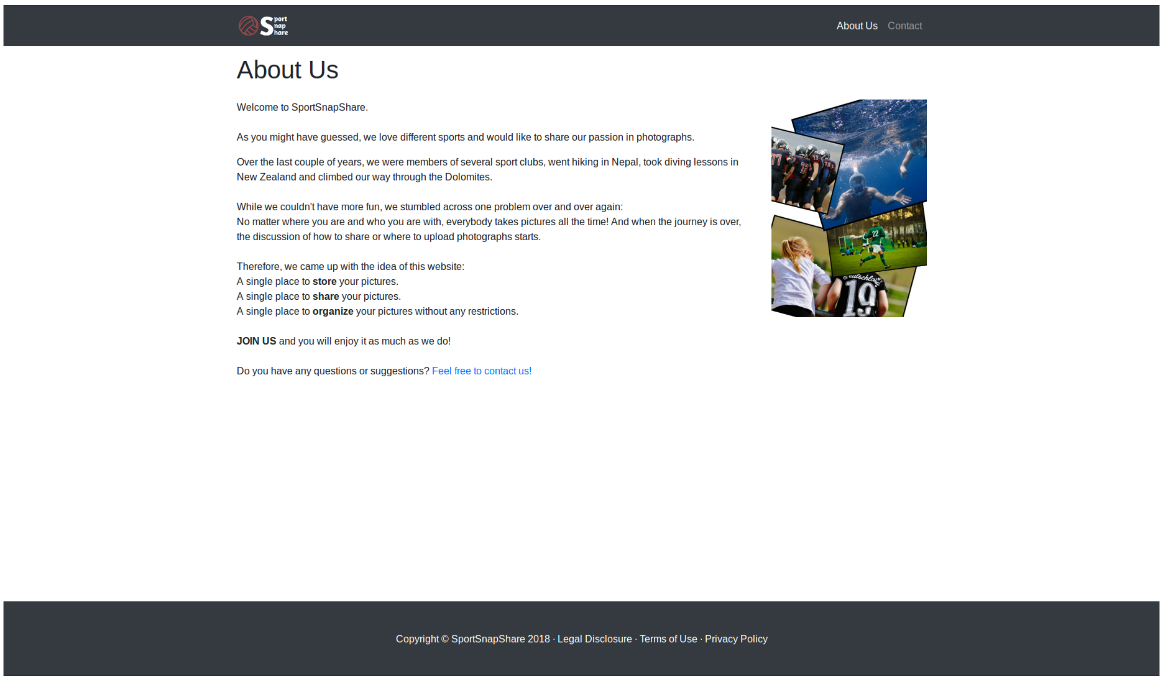


SportSnapShare Registration

Coming soon

Copyright © SportSnapShare 2018 · Legal Disclosure · Terms of Use · Privacy Policy

Figure C.2.: Registration Form



About Us

Welcome to SportSnapShare.

As you might have guessed, we love different sports and would like to share our passion in photographs.

Over the last couple of years, we were members of several sport clubs, went hiking in Nepal, took diving lessons in New Zealand and climbed our way through the Dolomites.

While we couldn't have more fun, we stumbled across one problem over and over again:
No matter where you are and who you are with, everybody takes pictures all the time! And when the journey is over, the discussion of how to share or where to upload photographs starts.

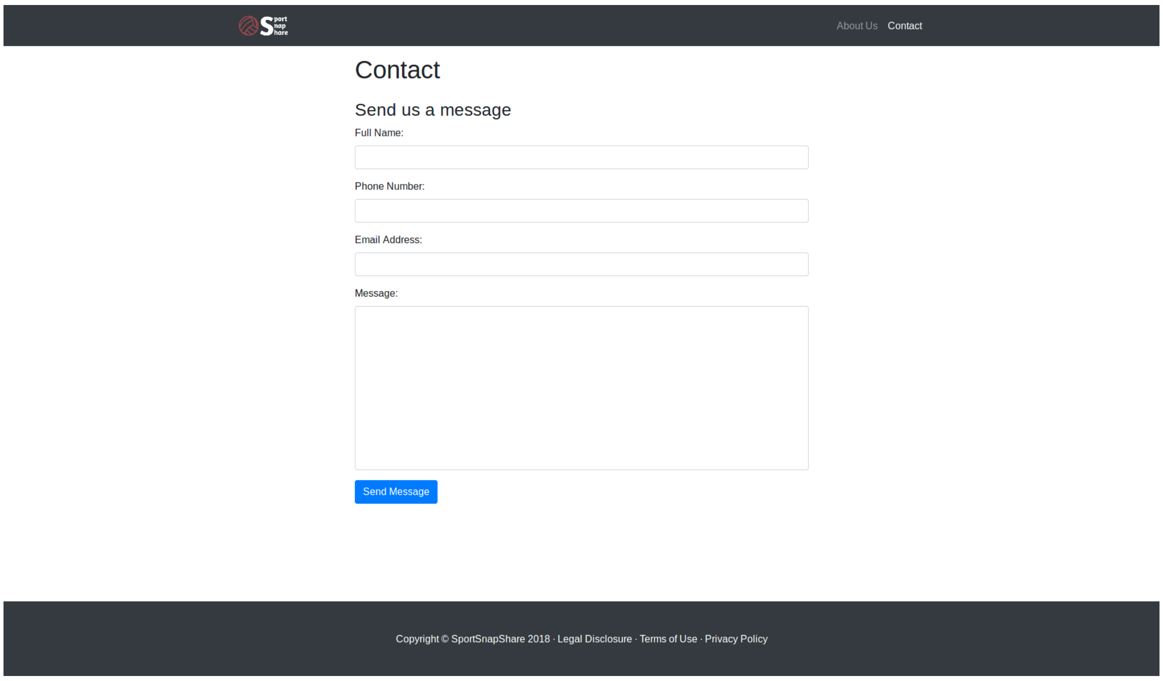
Therefore, we came up with the idea of this website:
A single place to **store** your pictures.
A single place to **share** your pictures.
A single place to **organize** your pictures without any restrictions.

JOIN US and you will enjoy it as much as we do!

Do you have any questions or suggestions? Feel free to contact us!

Copyright © SportSnapShare 2018 · Legal Disclosure · Terms of Use · Privacy Policy

Figure C.3.: About Us
Sport Photos: ©Alexandra [231], ©Teodor Andersson [232], ©Keith Johnston [233], ©Noz Urbina [234]



About Us Contact

Contact

Send us a message

Full Name:

Phone Number:

Email Address:

Message:

Copyright © SportSnapShare 2018 · Legal Disclosure · Terms of Use · Privacy Policy

Figure C.4.: Contact Form

Phase	*Category*	*Code*	*Participant*
Request	**Dependent Security & Requirements**	Security dependent on task	$N2_{100}$
		Security dependent on client requirements	$N6_{100}, N10_{100}, P9_{200}, P7_{100}$
		Security dependent on requirements	$P5_{100}, N11_{100}, N7_{100}$
		Security dependent on application	$N4_{100}$
		Task complexity	$N7_{100}$
		If you want security ask for it.	$P2_{200}, P7_{200}, N5_{200}$
		Meet all criteria.	$N11_{100},N9_{200}, N5_{200}$
	Security Awareness	Needed confirmation.	$N10_{200}$
		Do you need security?	$N10_{100}$
Implementation	**Misconceptions**	Password Security Misconception	$P5_{200}, N5_{100}, P1_{200}, P2_{100}, P6_{200}, N7_{100}, N9_{200}, N11_{100}, N6_{100}, P2_{200}, N10_{100}, P3_{200}, N12_{100}$
		Do not update knowledge	$N11_{100}, P2_{200}, N8_{100}, N10_{100}$
		Believes stored password securely	$N6_{100}, N10_{100}, P9_{200}, P7_{100}, N3_{100}, N1_{200}, P10_{200},P4_{200}, P1_{200}, P6_{200},N7_{100}, N12_{100}, N7_{200}, N9_{200}$
		Base64 is secure	$N3_{100}, N6_{100}, N7_{200}, P10_{200}, N12_{100}$
		Believe used optimal security	$N6_{100}, N1_{200}, N10_{100}, N12_{100}, N9_{200}$
		Threat model	$P10_{200}, N11_{100}, N5_{200}$
		Synonym hash encryption	$N10_{200}, N10_{100}, N5_{200},N4_{200},P7_{100},N3_{200},P5_{100},P8_{200},P4_{100},N8_{100},N2_{200},P4_{200},P3_{200},P9_{200}, N6_{200}$
	Functionality first	Task complexity	$N7_{100}$
		Security one separate part of application	$N3_{200}$
	Security costs extra	Security costs money	$N11_{100}$
		Security needs time	$P9_{100}$
	Information source	Easy documentation	$P4_{100}, P9_{200}$
		Examples and tutorials available	$N4_{100}$
		Documentation available	$N6_{200}, N8_{200}$
	Library usability	Available methods	$P5_{100}, N10_{200}, P11_{200}$
		Easy to use	$N1_{100}, N10_{100},N3_{200}, N6_{200}, P4_{200}$
		Few code	$P8_{100}, P11_{200}$
		Automatic security	$N10_{200}$
		Javax.crypto hard to use	$P2_{100}, N7_{100}$
Reflection	**Self-reflection**	Cocky	$N1_{100}, N4_{200}, P11_{200}$
		Missing knowledge	$P3_{100}, P7_{200}, N8_{200}$
		Missing knowledge of parameters	$N8_{200}$
		Aware solution is not best	$P3_{100}, P9_{100}, P10_{200}$
		Code can always be improved	$P10_{200}, N6_{100}$
		Nobody wants insecurity	$N6_{200}$
	Standards	Industry standard	$P8_{100}, P9_{200}$
		Standards	$N11_{100}, P3_{200}, N2_{100}, P2_{100}, N4_{100}, P12_{200}$
	Experience	Experienced no security issues	$P4_{200}, N9_{100}$
	Tests	Conducted tests	$N9_{100}, N9_{200}, P7_{200}$
	Trust in APIs	Trusts third parties	$P6_{100}, P9_{200}, N11_{100}, P11_{200}$
	Social desirability vs. reality	Indicated to be security aware (needed security hint)	$P6_{200}, N1_{100}, N1_{200}, N3_{200}, N8_{200}, N10_{100}, N10_{200}, N11_{100}, N9_{200}, P4_{100}, P6_{200}, N7_{100}, N8_{100}, N4_{200}$

Table C.1.: Coding of open questions

D. Appendix

D.1. Demographics

Table D.1 gives further detail about our participants' demographics.

D.2. Participant Submissions

Table D.2 shows an overview of participants' submissions we received as initial solutions, after the security request SecRequest-P and after the security request SecRequest-G per variable and per security request.

D.3. Qualitative Analysis

The coding overview is available in Table D.3.

D.4. Study Invitation

Software Developer Wanted for Study
Dear software developer,
The Institute of Computer Science of the University of Bonn is planning to conduct a scientific study to which I would like to cordially invite you. As expense allowance for the participation you receive 400 Euro.

Required knowledge: Java

Details: We are looking for motivated software developers / computer scientists who would like to participate in a scientific study on web development in Java. The goal is to test the usability of various Java APIs for web development. In the study you will be asked to implement parts of a web application and to participate in a subsequent survey. Java knowledge is required. The study is expected to take 5-6 hours to complete. You can work on the task whenever it fits you. With the study, we do not want to test you or your knowledge, but rather examine the usability of the APIs. Anonymized data are expected to be published in a scientific paper. Your consent will be asked again. The confidentiality of our participants' data is very important to us. All data collected in the study will be treated anonymously. The study is conducted independently of the company and no personal data will be exchanged with your company or third parties. At no time, neither in the study nor in a scientific publication, there will

Used IDE	Eclipse: 18 IntelliJ: 15 Visual Studio Code: 1
Profession (other)	IT-Consultant: 2 Testengineer: 2 System architect: 1 SoftwareConfigurationManager: 1
Field of study (university)	Computer Science: 17 Business informatics: 6 Electrical engineering: 3 Embedded System Design: 1 Operations Research: 1 Aerospace technology: 1 Mathematics: 1 Software Engineering: 1 Technical information: 1
Was taught IT-Sec at university?	Yes: 9 No: 15 I don’t recall: 5
Was taught IT-security outside of university?	Yes: 12, e.g. - Workshops and trainings - On the job - Youtube No: 17
Develops type(s) of software	Enterprise applications: 28 Web applications: 24 Desktop applications: 11 Mobile/App applications: 6
Company works in field of..	Web development: 7 Development of middleware, system components, libraries and frameworks: 5 Insurance: 6 Development of network and communication software : 2 Consulting: 2

Team Size	Min = 2, max = 50
	Mean = 13.1, median = 8
	sd = 11.8

Team has security focus	Yes: 7 No: 29
Works on safety-relevant tasks	Yes: 21 No: 15

Table D.1.: Details of demographics of 36 participants

	Initial solution				**After SecRequest-P**				**After SecRequest-G**			
	Non-secure	Secure	Score	Total	Non-secure	Secure	Score	Total	Non-secure	Secure	Score	Total
Non-Prompting	11	4	μ = 1.33 (σ = 2.35) min = 0, max = 6	15	2	9	μ = 3.5 (σ = 2.56) min = 0, max = 6	11	0	9	μ = 6.22 (σ = 1.64) min = 2 , max = 7	9
Prompting	1	20	μ = 5.36 (σ = 1.33) min = 0, max = 6	21	-	-	- -	-	0	8	μ = 6.25 (σ = 0.82) min = 5, max = 7	8
JSF	7	11	μ = 3.11 (σ = 2.61) min = 0, max = 6	18	2	4	μ = 1.92 (σ = 2.25) min = 0, max = 5	6	0	15	μ = 6.3 (σ =1.36) min = 2 , max = 7	15
Spring	5	13	μ = 4.25 σ = 2.73) min = 0, max = 6	18	0	5	μ = 5.4 (σ = 1.34) min = 3, max = 6	5	0	2	μ = 7 (σ =0) min = 7 , max = 7	2
Total	12	24	μ = 3.68 (σ = 2.69)	36	2	9	μ = 3.5 (σ = 2.56)	11	0	17	μ = 6.38 (σ = 1.29)	17

Table D.2.: Number of (non-)secure solutions and security score per variable and security request

be a connection drawn to the company or institution of the participant. As expense allowance for the participation in the study you get 400 Euro.

Interested? Then please fill in the following questionnaire: LINK

The expression of interest does not guarantee participation. The study will be conducted in cooperation with the working group of the University of Bonn. The expense allowance will be compensated by the University of Bonn and is independent of your company.

D.5. Playbook

We used the same playbook as used in the study of Naiakshina et al. [117]. We had to extend and adapt it by several relevant aspects for our sample, which can be found in the following. *P* indicates the participant and *R* indicates the researcher.

Study Offer

After participants filled out the questionnaire which indicated their interest, they were contacted via email:

- 1. Message: *R: Hello, thank you for your interest in our study! Before we can start, you need to give your consent. To do so, please visit the following website: LINK. Your study-ID is: STUDY-ID. Please use this study-ID every time you get asked for it. Thank you! Kind regards, ..*
- 2. Message: *R: Hello, you gave you consent to take part in our study. Thank you very much! You can now download the task including the code as ZIP file from here: LINK. Everything you need to know will be explained in the task description. After you completed the task, please upload your solution. Other participants are working on different tasks. I would kindly ask you not to communicate about the task until after we sent you the study results. Otherwise, this could potentially falsify the results. Thank you very much and enjoy coding. Kind regards, ...*

Deadline

- 10 days after task was sent and in case no deadline was set: *R*: Hi, could you give us a status update? What do you think how long will you need to complete the task?
- If no reaction after 3 weeks: *R*: Hello, please sent your solution till *date in one week*. If you decided to no longer participate in the study, I would be very glad if you could let me know. Thank you and kind regards, ...

Password-related Questions

- Participant handed in insecure code: *R*: I saw that the password is stored in clear text. Could you also store it securely?

- Password score < 6: *R:* Thank you for submitting your solution. Now I have one further request. I noticed, that you did not follow industry best practices, e.g., NIST(National Institute of Standards and Technology) or OWASP (Open Web Application Security Project), to securely store the end-user password. Could you please revise your submission and ensure that you follow industry best practices? You can find some information on OWASP on this website: `https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Password_Storage_Cheat_Sheet.md` and information on NIST on this website: `https://pages.nist.gov/800-63-3/sp800-63b.html` in section 5.1.1.2.

General Questions

- *P*: Do you use Skype? *R*: No, not for this study, sorry!
- *P*: How important are Java skills? *R*: The programming task will be in Java and we assume 2 years of Java experience to be necessary to solve it purposefully.
- *P*: What exactly do you mean by “The study is expected to take 5-6 hours to complete.”? *R*: Without disruption, participants in previous studies needed approx. 5 to 6 hours to complete the task.
- *P*: Is there an on-site part? *R*: No, it is an online study.
- Participant is confused after SecRequest-G: *R*: You will not receive any further request. You can choose, which industry standard you would like to follow; OWASP or NIST. So that you do not have to read the whole NIST guideline, you can read all necessary information in section 5.1.1.2. This section approximately complies with the length of the provided OWASP source. It is up to you to choose one standard. Afterwards you only have to fill out a subsequent survey, which concludes the study.

Implementation and Technical Questions

- *P*: I noticed that the database server does not seem to be running. *R*: I checked it, the database server is running. Errors might arise, because the code is not complete yet.

Survey Request

Hello, thank you for your solution! As announced in the invitation, we would like to invite you to participate in a subsequent survey. Following the survey you will receive instructions on how to receive your compensation. Please make sure to fill out your data on your computer to ensure readability. Please find the survey here: LINK Best regards, ...

D.6. Survey for Company Developers

We used a seven-point rating scale according to [194]. The italic text indicates the answer possibilities.

D.6.1. Pre-Screening

1. What is your current main occupation? *Employee/Freelancer/Researcher/Apprentice/Undergraduate part-time student/Undergraduate full-time student/Graduate part-time student/Graduate full-time student/Other (please specify)*
2. In which field are you currently working in your company? *Consulting/Software development/Testing/Other (please specify)*
3. How many years of experience do you have with software development in general? *[Number]*
4. How many years of experience do you have with Java development? *[Number]*
5. Is software development part of your current job? *Yes/No/Other (please specify)*

D.6.2. Exit Survey

The following questions were asked after the programming task was completed.

1. Your study-ID:
2. Which IDE did you use to solve the task?
3. How familiar are you with Java?
   1 - Not familiar at all - 7 - Very familiar
4. How familiar are you with PostgreSQL?
   1 - Not familiar at all - 7 - Very familiar
5. How familiar are you with Hibernate?
   1 - Not familiar at all - 7 - Very familiar
6. How familiar are you with (Q2)?
   1 - Not familiar at all - 7 - Very familiar
7. How familiar are you with JavaServerFaces/Spring?
   1 - Not familiar at all - 7 - Very familiar
8. How long have you been actively working on the task to solve it? Your answer has no influence on the payment but only serves as an estimate for the workload. [Please indicate the time in full hours.]
   [Number]
9. Overall, this task was ...?
   1 - Very Difficult - 7 - Very Easy
10. How familiar are you with processing the assigned task?
    1 - Not familiar at all - 7 - Very familiar
11. How close was the task to reality compared to the projects that you develop in everyday life?
    1 - Not close at all - 7 - Very close
12. Did you already gain experience in the past with the assigned task?
    No / Yes, at the university / Yes, on the job / Yes, other: [Free text]
13. Do you think your solution is optimal?
    Yes / No

  - Why do you think your solution is (not) optimal?
    [Free text]

14. I have a good understanding of security concepts.
    1 - Strongly disagree - 7 - Strongly agree
15. How often do you ask for help when faced with security problems?
    1 - Never - 7 - Every time
16. How often are you asked for help when others are faced with security problems?
    1 - Never - 7 - Every time
17. How often do you need to add security to the software you develop in general (apart from this study)?
    1 - Never - 7 - Every time
18. How often have you stored passwords in a database in the software you have developed (apart from this study)?
    [Free text]
19. How would you rate your background/knowledge with regard to secure password storage in a database?
    1 - Not knowledgeable at all - 7 - Very knowledgeable
20. Do you think that you stored the end-user passwords securely?
    Yes / No
    - If yes:
      * What did you do to store the passwords securely?
        [Free text]
      * Do you think your solution is optimal with regard to security?
        Yes / No
        · Why do you think your solution is (not) optimal with regard to security?
        [Free text]
      * Do you think you would have stored end-user passwords securely, if you had not been told about it?
        Yes / No
      * Please explain your decision.
        [Free text]
    - If no:
      * Why do you think that you did not store the passwords securely?
        [Free text]
      * What would you do, if you needed to store the end-user password securely?
        [Free text]
      * In the previous questions, you mentioned why your solution is not secure and what you would need to do to obtain secure password storage. Why did you not implement your suggestion?
        [Free text]
21. If you had the same job outside of a study, would you have solved it differently in terms of secure password storage?
    Yes / No

22. Please explain your decision.
    [Free text]
23. Did you use libraries to store the end-user passwords securely?
    Yes / No
    - If yes:
        * Which libraries did you use to store the end-user passwords securely (in this study)?
          [Free text]
        * Please name the most relevant library you have used to store the end-user passwords securely (in this study).
          [Free text]
        * You have identified *** as the most relevant library to store end-user passwords securely. How would you rate its ease of use in terms of accomplishing your tasks functionally?
          1 - Very difficult - 7 - Very easy
        * Please explain your decision.
          [Free text]
        * You have identified *** as the most relevant library to store end-user passwords securely. How would you rate its ease of use in terms of accomplishing your tasks securely?
          1 - Very difficult - 7 - Very easy
        * Please explain your decision.
          [Free text]
24. JSF/ Spring: Usability scale (see Appendix B.2.1); the term library was replaced by framework
25. Have you used Java APIs/libraries to store end-user passwords securely in a database before?
    Yes / No
    - If yes:
        * Which Java APIs/libraries to store end-user passwords securely in a database have you used before?
          [Free text]
        * What is your most-used API/library for secure password storage in a database?
          [Free text]
        * How would you rate its ease of use in terms of accomplishing your tasks functionally?
          1 - Very difficult - 7 - Very easy
        * Please explain your decision.
          [Free text]
        * How would you rate its ease of use in terms of accomplishing your tasks securely?
          1 - Very difficult - 7 - Very easy
        * Please explain your decision.
          [Free text]
26. Have you ever heard of NIST (National Institute of Standards and Technology) or OWASP (Open Web Application Security Project)?
    Yes / No

- If yes:
  - Did you follow one of these practices programming our task?
    Yes / No
    If yes:
    - Which of these practices did you follow?
      NIST / OWASP
    - Please explain your decision.
      [Free text]

    If no:
    - Why did you not follow one of these practices?
      [Free text]

D.6.3. Demographics

1. Please select your gender.
   Female / Male / Prefer not to say / Diverse: [Free text]
2. Age
   [Number]
3. What is your current main occupation?
   Freelance developer / Industrial developer / Industrial researcher / Academic researcher / Undergraduate part-time student / Undergraduate full-time student / Graduate part-time student / Graduate full-time student / Other: [Free text]
4. Do you have a university degree?
   Yes / No
   If yes:
   - At which university/universities were/are you enrolled?
     [Free text]
   - What was/is your subject?
     [Free text]
   - Were/Are you taught about IT-security at university?
     Yes / No / I don't recall
     - If yes:
       - Which security lectures did you pass in your Bachelor/Master programme?
         [Free text]
       - Was at least one of these modules a compulsary module?
         Yes / No / I don't recall
   - Were/Are you taught about IT-security apart from university?
     Yes / No
     - If yes:

      · Where were/are you taught about IT-security apart from university?
        [Free text]

   - What was your main source of learning about IT-security?
     [Free text]

   If no:

   - How did you gain your IT skills?
     [Free text]

   - How did you gain your IT-security skills?
     [Free text]

5. What type(s) of software do you develop? (Multiple answers possible)
   Web applications / Mobile/App applications / Desktop applications / Embedded Software Engineering / Enterprise applications / Other: [Free text]

6. How many years of experience do you have with software development in general?
   [Number]

7. How many years of experience do you have with Java development?
   [Number]

8. What is your nationality?
   [Free text]

9. How old is your organisation? [Please specify in years.]
   [Number]

10. What is the total number of employees in your organization?
    1-9 / 10-249 / 250-499 / 500-999 / 1000 or more

11. How many members are there in your team?
    [Number]

12. Which field of activity does your company belong to?
    Game development / Development of network and communication software / Web development / Development of middleware, system components, libraries and frameworks / Development of other tools for developers, such as IDEs and compilers / Other: [Free text]

13. Does your company have a security focus?
    Yes / No

14. Does your team have a security focus in its current field of activity?
    Yes / No / I work alone and my field of activity has a security focus. / I work alone and my field of activity has no security focus.

15. Do you also have to work on security-relevant tasks in your field of activity?
    Yes / No

16. How do you rate the payment of the study?
    Way too little / Too little / Just right / Too much / Way too much

17. Do you wish to be contacted by our research group to be informed about the study results?
    Yes / No

18. Would you like to be invited by our research group for further studies in the future?
    Yes / No
19. If you wish to be contacted by our research group, please provide us your email address. The email address will be stored separately from the study data. Email address:
    [Free Text]

Category	*Code*	*Participant*
Experience	Experience	NS5
	Knowledge mismatch	PJ5
	No experience	PJ3, PJ4, PJ6, PJ10, NJ1, NJ3, NJ7, NJ6, NS1
Insecurity	Continuous learning	PJ5, PS8
	Insecurity	PS6, NS3
	Internet search & outdated information	PJ6, PS8
	Lack of knowledge	PJ7, NJ8
	Security is not part of everyday development	PJ5, PS8
NIST	NIST easy	PJ4
	NIST clear requirements	NJ2
	NIST not trusted because of US origin	PJ5, PS3
OWASP	OWASP familiarity	NJ3, NJ7, PS7
	OWASP important for web security	PS9
	OWASP open source	PJ3
	OWASP practical example	PJ5, PJ8, NJ3, NJ6, NS4
OWASP & NIST need of improvement	Inconsistent information	PJ5
	NIST difficult	PJ5
	Too complicated	PJ5
Requirements	Depend on Requirements	PJ1, PJ10, NJ2, NJ3, NJ4, PS6, NS4, NS7
	OWASP/NIST security fulfilled	PJ7, PJ9, NJ5, NS4
	Quality depends on requirements	PS1
	Requirements fulfilled	PJ9, PS10
	Security depends on requirements	PJ1, PJ6, NJ2, NJ3, NJ4, NJ6, PS8, PS1, NS2
Security entity	Security consultant	PJ10, NJ2, NJ7, PS5, PS8
	Security experienced collegue	PS6, NS1
Trade-off between effort & requirements	Always space for improvement	NS1
	Does not use technologies daily	PS3
	Effort vs. benefit	NJ6
	Trade-off between effort & requirements	NJ4, NJ5, NJ6, NJ7
Trust in standards	State-of-the-art security	PJ10, NJ1
	Trust in API	PS1, PS2, PS5, PS7, PS10, NS5, NS7
	Trust in Bcrypt	PS7, NS3, NS5, NS7
	Trust in NIST	PJ9
	Trust in OWASP	PJ10
	Would not implement security by herself/himself	PS5, PS11

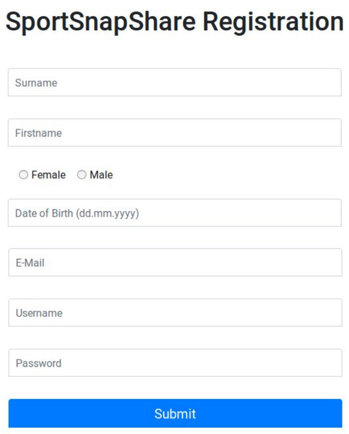
Table D.3.: Coding of open questions

D.7. Non-prompted/Prompted Task

Welcome! *We would like to invite you to participate in our study and thus make a valuable contribution to our research. Our research goal is to study the usability of the Web Framework* ***Java Server Faces (JSF). We are specially interested in the security aspects of password storage with JSF.***

Thank you so much for helping us finishing our project!

We are developing a social networking website to share pictures with family and friends. People need to register (email, name, gender, birth date, username and password) to this website in order to share their pictures. As we already wrote in the project description, we need help with finishing the registration logic for our social networking website. You can find our website here: LINK



The front-end is already developed. Another developer started to implement the program logic (MVC), but left the team recently. **We need you to complete the partially available implementation by programming the registration functionality in the back-end. (Prompting: Please ensure that the user password is stored securely.)**

According to our former developer, the following things have to be done:

<u>Setting up the environment:</u>
The following software will be needed:
1. Tomcat server
2. Optional: Eclipse IDE for Java EE Developers
3. Optional: pgAdmin

<u>Prepare the project & the database:</u>
If you use Eclipse, it could look the following way:

1. Import the existing project as Existing Maven Project.
2. Add Maven Dependencies to the project in Eclipse: Right mouse click on the project/Properties/Deployment Assembly/Add/Java Build Path Entries → choose Maven Dependencies → Finish→ Apply

3. Connect to database (For example in pgAdmin: File/Add Server…)
You will find all necessary information (host, port, DB Name, username and password) in the *hibernate.cfg.xml* file (src/main/resources).

<u>Implementing the registration process:</u>
This was our developer's TODO list, it might not be complete, but hopefully it will help you understand what already was developed and what is left to be done. Figure 1 shows how the different parts are communicating with each other.

In order to test/start your application: Right mouse click on the project/Run As/Run on Server

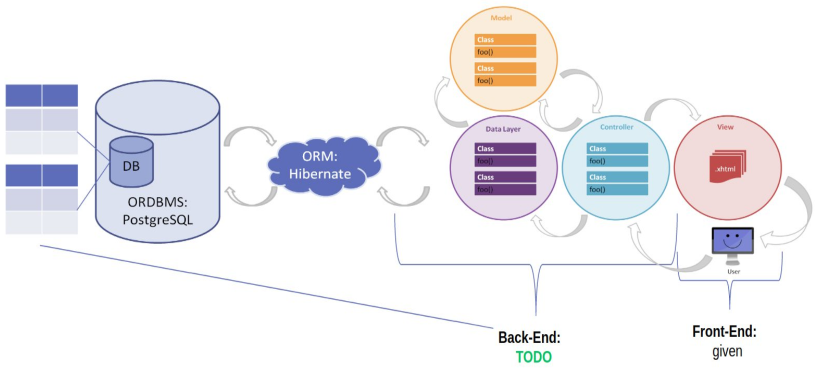


Figure 1

Util:

1. Done: Hibernate configuration
2. Done: Implement method: getSessionFactory()

View:

1. Done: Create registration.xhtml
2. Done: Create successRegistration.xhtml

Database:

1. TODO: Add table *appuser* in the database

Model – Appuser.java

1. TODO: Add user properties (see registration.xhtml)
2. TODO: Add getters & setters

DAO – UserDAO.java

1. Done: Implement mehtod: validateUsername()
2. Done: Open session
3. TODO: Implement method: save()

Controller – AppController.java

1. TODO: Add variables for properties
2. TODO: implement method: saveUser()

(Prompting**:** Please ensure that the **user password is stored securely.)**

Bibliography

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

[2] Matthew Green and Matthew Smith. Developers are not the enemy!: The need for usable security apis. *IEEE Security & Privacy*, 14(5):40–46, 2016.

[3] Mary Ellen Zurko. User-centered security: Stepping up to the grand challenge. In *Proceedings of the 21st Annual Computer Security Applications Conference*, ACSAC '05, pages 187–202, Washington, DC, USA, 2005. IEEE Computer Society.

[4] Alma Whitten and J Doug Tygar. Why johnny can't encrypt: A usability evaluation of pgp 5.0. In *Usenix Security*, volume 1999, 1999.

[5] Steve Sheng, Levi Broderick, Colleen Alison Koranda, and Jeremy J Hyland. Why johnny still can't encrypt: evaluating the usability of email encryption software. In *Symposium On Usable Privacy and Security*, pages 3–4. ACM, 2006.

[6] Sandy Clark, Travis Goodspeed, Perry Metzger, Zachary Wasserman, Kevin Xu, and Matt Blaze. Why (special agent) johnny (still) can't encrypt: A security analysis of the apco project 25 two-way radio system. In *USENIX Security Symposium*, volume 2011, pages 8–12, 2011.

[7] Sascha Fahl, Marian Harbach, Thomas Muders, Matthew Smith, and Uwe Sander. Helping johnny 2.0 to encrypt his facebook conversations. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, pages 1–17, 2012.

[8] Karen Renaud, Melanie Volkamer, and Arne Renkema-Padmos. Why doesn't jane protect her privacy? In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 244–262. Springer, 2014.

[9] Ruba Abu-Salma, M Angela Sasse, Joseph Bonneau, Anastasia Danilova, Alena Naiakshina, and Matthew Smith. Obstacles to the adoption of secure communication tools. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 137–153, Piscataway, NJ, USA, 2017. IEEE, IEEE Press.

[10] Sergej Dechand, Alena Naiakshina, Anastasia Danilova, and Matthew Smith. In encryption we don't trust: The effect of end-to-end encryption to the masses on user perception. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 401–415. IEEE, 2019.

[11] Fahimeh Raja, Kirstie Hawkey, Steven Hsu, Kai-Le Clement Wang, and Konstantin Beznosov. A brick wall, a locked door, and a bandit: A physical security metaphor for firewall warnings. In *Proceedings*

of the Seventh Symposium on Usable Privacy and Security, SOUPS'11, New York, NY, USA, 2011. Association for Computing Machinery.

[12] Jialiu Lin, Shahriyar Amini, Jason I Hong, Norman Sadeh, Janne Lindqvist, and Joy Zhang. Expectation and purpose: understanding users' mental models of mobile app privacy through crowdsourcing. In *Proceedings of the 2012 ACM conference on ubiquitous computing*, pages 501–510, 2012.

[13] Rick Wash. Folk models of home computer security. In *Proceedings of the Sixth Symposium on Usable Privacy and Security*, pages 1–16, 2010.

[14] Kami Vaniea, Emilee Rader, and Rick Wash. Mental models of software updates. *International Communication Association*, 2014.

[15] Fahimeh Raja, Kirstie Hawkey, and Konstantin Beznosov. Revealing hidden context: improving mental models of personal firewall users. In *Proceedings of the 5th Symposium on Usable Privacy and Security*, pages 1–12, 2009.

[16] Farzaneh Asgharpour, Debin Liu, and L Jean Camp. Mental models of security risks. In *International Conference on Financial Cryptography and Data Security*, pages 367–377. Springer, 2007.

[17] Cristian Bravo-Lillo, Lorrie Faith Cranor, Julie Downs, and Saranga Komanduri. Bridging the gap in computer security warnings: A mental model approach. *IEEE Security & Privacy*, 9(2):18–26, 2010.

[18] Marian Harbach, Markus Hettig, Susanne Weber, and Matthew Smith. Using personal examples to improve risk communication for security & privacy decisions. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 2647–2656, 2014.

[19] Joshua Sunshine, Serge Egelman, Hazim Almuhimedi, Neha Atri, and Lorrie Faith Cranor. Crying wolf: An empirical study of SSL warning effectiveness. In *USENIX security symposium*, pages 399–416. Montreal, Canada, 2009.

[20] Katharina Krombholz, Karoline Busse, Katharina Pfeffer, Matthew Smith, and Emanuel von Zezschwitz. "If HTTPS Were Secure, I Wouldn't Need 2FA"-End User and Administrator Mental Models of HTTPS. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 246–263. IEEE, 2019.

[21] Adam Beautement, M Angela Sasse, and Mike Wonham. The compliance budget: managing security behaviour in organisations. In *Proceedings of the 2008 New Security Paradigms Workshop*, pages 47–58, 2008.

[22] Johannes Sänger, Norman Hänsch, Brian Glass, Zinaida Benenson, Robert Landwirth, and M Angela Sasse. Look before you leap: improving the users' ability to detect fraud in electronic marketplaces. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 3870–3882, 2016.

[23] Karla Badillo-Urquiola, Xinru Page, and Pamela J Wisniewski. Risk vs. restriction: The tension between providing a sense of normalcy and keeping foster teens safe online. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–14, 2019.

[24] Katharina Krombholz, Adrian Dabrowski, Matthew Smith, and Edgar Weippl. Exploring design directions for wearable privacy. 2017.

[25] Katharina Krombholz, Aljosha Judmayer, Matthias Gusenbauer, and Edgar Weippl. The other side of the coin: User experiences with bitcoin security and privacy. In *International conference on financial cryptography and data security*, pages 555–580. Springer, 2016.

[26] Kelsey R Fulton, Rebecca Gelles, Alexandra McKay, Yasmin Abdi, Richard Roberts, and Michelle L Mazurek. The effect of entertainment media on mental models of computer security. In *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*, 2019.

[27] Claire Dolin, Ben Weinshel, Shawn Shan, Chang Min Hahn, Euirim Choi, Michelle L Mazurek, and Blase Ur. Unpacking perceptions of data-driven inferences underlying online targeting and personalization. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2018.

[28] Wei Bai, Ciara Lynton, Charalampos Papamanthou, and Michelle L Mazurek. Understanding user tradeoffs for search in encrypted communication. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 258–272. IEEE, 2018.

[29] Daniel Votipka, Seth M Rabin, Kristopher Micinski, Thomas Gilray, Michelle L Mazurek, and Jeffrey S Foster. User comfort with android background resource accesses in different contexts. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, pages 235–250, 2018.

[30] Kristopher Micinski, Daniel Votipka, Rock Stevens, Nikolaos Kofinas, Michelle L Mazurek, and Jeffrey S Foster. User interactions and permission use on android. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 362–373, 2017.

[31] Elissa M Redmiles, Sean Kross, and Michelle L Mazurek. Where is the digital divide? a survey of security, privacy, and socioeconomics. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 931–936, 2017.

[32] Ben Weinshel, Miranda Wei, Mainack Mondal, Euirim Choi, Shawn Shan, Claire Dolin, Michelle L Mazurek, and Blase Ur. Oh, the places you've been! user reactions to longitudinal transparency about third-party web tracking and inferencing. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 149–166, 2019.

[33] Angelisa C Plane, Elissa M Redmiles, Michelle L Mazurek, and Michael Carl Tschantz. Exploring user perceptions of discrimination in online targeted advertising. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 935–951, 2017.

[34] Wei Bai, Moses Namara, Yichen Qian, Patrick Gage Kelley, Michelle L Mazurek, and Doowon Kim. An inconvenient trust: User attitudes toward security and usability tradeoffs for key-directory encryption systems. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, pages 113–130, 2016.

[35] Michelle L Mazurek, Peter F Klemperer, Richard Shay, Hassan Takabi, Lujo Bauer, and Lorrie Faith Cranor. Exploring reactive access control. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2085–2094, 2011.

[36] Michelle L Mazurek, JP Arsenault, Joanna Bresee, Nitin Gupta, Iulia Ion, Christina Johns, Daniel Lee, Yuan Liang, Jenny Olsen, Brandon Salmon, et al. Access control for home data sharing: Attitudes, needs and practices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 645–654, 2010.

[37] Hala Assal, Stephanie Hurtado, Ahsan Imran, and Sonia Chiasson. What's the deal with privacy apps? a comprehensive exploration of user perception and usability. In *Proceedings of the 14th International Conference on Mobile and Ubiquitous Multimedia*, pages 25–36, 2015.

[38] Farah Chanchary and Sonia Chiasson. User perceptions of sharing, advertising, and tracking. In *Eleventh Symposium On Usable Privacy and Security (SOUPS 2015)*, pages 53–67, 2015.

[39] Hana Habib, Jessica Colnago, Vidya Gopalakrishnan, Sarah Pearman, Jeremy Thomas, Alessandro Acquisti, Nicolas Christin, and Lorrie Faith Cranor. Away from prying eyes: analyzing usage and understanding of private browsing. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, pages 159–175, 2018.

[40] Alain Forget, Sarah Pearman, Jeremy Thomas, Alessandro Acquisti, Nicolas Christin, Lorrie Faith Cranor, Serge Egelman, Marian Harbach, and Rahul Telang. Do or do not, there is no try: user engagement may not improve security outcomes. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, pages 97–111, 2016.

[41] Christian Tiefenau, Emanuel von Zezschwitz, Maximilian Häring, Katharina Krombholz, and Matthew Smith. A usability evaluation of let's encrypt and certbot: Usable security done right. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1971–1988, 2019.

[42] Constanze Dietrich, Katharina Krombholz, Kevin Borgolte, and Tobias Fiebig. Investigating system operators' perspective on security misconfigurations. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1272–1289, 2018.

[43] Katharina Krombholz, Wilfried Mayer, Martin Schmiedecker, and Edgar Weippl. "I have no idea what i'm doing" - on the usability of deploying HTTPS. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1339–1356, Vancouver, BC, 2017. USENIX Association.

[44] Sascha Fahl, Yasemin Acar, Henning Perl, and Matthew Smith. Why eve and mallory (also) love webmasters: a study on the root causes of ssl misconfigurations. In *Proceedings of the 9th ACM symposium on Information, computer and communications security*, pages 507–512, 2014.

[45] Nicole F Velasquez and Suzanne P Weisband. Work practices of system administrators: implications for tool design. In *Proceedings of the 2nd ACM Symposium on Computer Human Interaction for Management of Information Technology*, pages 1–10, 2008.

[46] Matthew Bernhard, Jonathan Sharman, Claudia Ziegler Acemyan, Philip Kortum, Dan S Wallach, and J Alex Halderman. On the usability of https deployment. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–10, 2019.

[47] David Botta, Rodrigo Werlinger, André Gagné, Konstantin Beznosov, Lee Iverson, Sidney Fels, and Brian Fisher. Towards understanding it security professionals and their tools. In *Proceedings of the 3rd symposium on Usable privacy and security*, pages 100–111, 2007.

[48] Peter Leo Gorski, Luigi Lo Iacono, Dominik Wermke, Christian Stransky, Sebastian Möller, Yasemin Acar, and Sascha Fahl. 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)*, pages 265–281, Baltimore, MD, 2018. USENIX Association.

[49] Yasemin Acar, Christian Stransky, Dominik Wermke, Michelle L. Mazurek, and Sascha Fahl. Security developer studies with github users: Exploring a convenience sample. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*, pages 81–95, Santa Clara, CA, 2017. USENIX Association.

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

[51] Sascha Fahl, Marian Harbach, Henning Perl, Markus Koetter, and Matthew Smith. Rethinking ssl development in an appified world. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 49–60. ACM, 2013.

[52] Lutz Prechelt. Plat_forms: A web development platform comparison by an exploratory experiment searching for emergent platform properties. *IEEE Transactions on Software Engineering*, 37(1):95–108, Jan 2011.

[53] Hala Assal and Sonia Chiasson. '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, pages 289:1–289:13, New York, NY, USA, 2019. ACM.

[54] Khaled Yakdan, Sergej Dechand, Elmar Gerhards-Padilla, and Matthew Smith. Helping johnny to analyze malware: A usability-optimized decompiler and malware analysis user study. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 158–177, San Jose, CA, USA, 2016. IEEE, IEEE.

[55] Richard Shay, Saranga Komanduri, Patrick Gage Kelley, Pedro Giovanni Leon, Michelle L. Mazurek, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Encountering stronger password requirements: User attitudes and behaviors. In *Proceedings of the Sixth Symposium on Usable Privacy and Security*, SOUPS '10, pages 2:1–2:20, New York, NY, USA, 2010. ACM.

[56] Saranga Komanduri, Richard Shay, Patrick Gage Kelley, Michelle L Mazurek, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, and Serge Egelman. Of passwords and people: Measuring the effect of password-composition policies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2595–2604, New York, NY, USA, 2011. ACM, ACM.

[57] Michelle L. Mazurek, Saranga Komanduri, Timothy Vidas, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, Patrick Gage Kelley, Richard Shay, and Blase Ur. Measuring password guessability for an entire university. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 173–186, New York, NY, USA, 2013. ACM.

[58] J. Bonneau. The science of guessing: Analyzing an anonymized corpus of 70 million passwords. In *2012 IEEE Symposium on Security and Privacy*, pages 538–552, San Francisco, CA, USA, May 2012. IEEE.

[59] Matt Weir, Sudhir Aggarwal, Michael Collins, and Henry Stern. Testing metrics for password creation policies by attacking large sets of revealed passwords. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 162–175. ACM, 2010.

[60] Blase Ur, Jonathan Bees, Sean M. Segreti, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Do users' perceptions of password security match reality? In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, pages 3748–3760, New York, NY, USA, 2016. ACM.

[61] 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. Designing password policies for strength and usability. *ACM Transactions on Information and System Security (TISSEC)*, 18(4):13:1–13:34, May 2016.

[62] Blase Ur, Fumiko Noma, Jonathan Bees, Sean M. Segreti, Richard Shay, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. 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)*, pages 123–140, Ottawa, 2015. USENIX Association.

[63] 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. Can long passwords be secure and usable? In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, CHI '14, pages 2927–2936, New York, NY, USA, 2014. ACM.

[64] 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. Helping users create better passwords. *USENIX*, 37(6):51–57, 2012.

[65] Blase Ur, Felicia Alfieri, Maung Aung, Lujo Bauer, Nicolas Christin, Jessica Colnago, Lorrie Faith Cranor, Henry Dixon, Pardis Emami Naeini, Hana Habib, and et al. Design and evaluation of a data-driven password meter. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, page 3775–3786, New York, NY, USA, 2017. Association for Computing Machinery.

[66] Serge Egelman, Andreas Sotirakopoulos, Ildar Muslukhov, Konstantin Beznosov, and Cormac Herley. 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*, pages 2379–2388, New York, NY, USA, 2013. ACM, ACM.

[67] M Angela Sasse, Michelle Steves, Kat Krol, and Dana Chisnell. The great authentication fatigue–and how to overcome it. In *International Conference on Cross-Cultural Design*, pages 228–239. Springer, 2014.

[68] Kat Krol, Eleni Philippou, Emiliano De Cristofaro, and M Angela Sasse. "They brought in the horrible key ring thing!" analysing the usability of two-factor authentication in uk online banking. *arXiv preprint arXiv:1501.04434*, 2015.

[69] Kurt Thomas, Jennifer Pullman, Kevin Yeo, Ananth Raghunathan, Patrick Gage Kelley, Luca Invernizzi, Borbala Benko, Tadek Pietraszek, Sarvar Patel, Dan Boneh, et al. Protecting accounts from credential stuffing with password breach alerting. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1556–1571, 2019.

[70] Naomi Woods and Mikko Siponen. Improving password memorability, while not inconveniencing the user. *International Journal of Human-Computer Studies*, 128:61–71, 2019.

[71] Sascha Fahl, Marian Harbach, Yasemin Acar, and Matthew Smith. On the ecological validity of a password study. In *Proceedings of the Ninth Symposium on Usable Privacy and Security*, page 13. ACM, 2013.

[72] Katharina Krombholz, Adrian Dabrowski, Peter Purgathofer, and Edgar Weippl. Poster: The petri dish attack-guessing secrets based on bacterial growth. In *Proceedings of the NDSS Symposium*, 2018.

[73] Katharina Krombholz, Thomas Hupperich, and Thorsten Holz. Use the force: Evaluating force-sensitive authentication for mobile devices. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, pages 207–219, 2016.

[74] Elissa M Redmiles, John P Dickerson, Krishna P Gummadi, and Michelle L Mazurek. Equitable security: Optimizing distribution of nudges and resources. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 2270–2272, 2018.

[75] Sean M Segreti, William Melicher, Saranga Komanduri, Darya Melicher, Richard Shay, Blase Ur, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, and Michelle L Mazurek. Diversify to survive: Making passwords stronger with adaptive policies. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS)*, pages 1–12, Santa Clara, CA, USA, 2017. USENIX Association.

[76] Elissa M Redmiles, Everest Liu, and Michelle L Mazurek. You want me to do what? a design study of two-factor authentication messages. In *SOUPS*, 2017.

[77] William Melicher, Darya Kurilova, Sean M Segreti, Pranshu Kalvani, Richard Shay, Blase Ur, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, and Michelle L Mazurek. Usability and security of text passwords on mobile devices. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 527–539, 2016.

[78] Richard Shay, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, Alain Forget, Saranga Komanduri, Michelle L Mazurek, William Melicher, Sean M Segreti, and Blase Ur. A spoonful of sugar? the impact of guidance and feedback on password-creation behavior. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 2903–2912, 2015.

[79] Richard Shay, Patrick Gage Kelley, Saranga Komanduri, Michelle L Mazurek, Blase Ur, Timothy Vidas, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Correct horse battery staple: Exploring the usability of system-assigned passphrases. In *Proceedings of the eighth symposium on usable privacy and security*, pages 1–20, 2012.

[80] Blase Ur, Patrick Gage Kelley, Saranga Komanduri, Joel Lee, Michael Maass, Michelle L Mazurek, Timothy Passaro, Richard Shay, Timothy Vidas, Lujo Bauer, et al. How does your password measure up? the effect of strength meters on password creation. In *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, pages 65–80, 2012.

[81] Sonia Chiasson, Paul C van Oorschot, and Robert Biddle. A usability study and critique of two password managers. In *USENIX Security Symposium*, volume 15, pages 1–16, 2006.

[82] Sonia Chiasson, Paul C Van Oorschot, and Robert Biddle. Graphical password authentication using cued click points. In *European Symposium on Research in Computer Security*, pages 359–374. Springer, 2007.

[83] Sonia Chiasson, Alain Forget, Elizabeth Stobert, Paul C Van Oorschot, and Robert Biddle. Multiple password interference in text passwords and click-based graphical passwords. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 500–511, 2009.

[84] Sonia Chiasson, Robert Biddle, and Paul C Van Oorschot. A second look at the usability of click-based graphical passwords. In *Proceedings of the 3rd symposium on Usable privacy and security*, pages 1–12, 2007.

[85] Sonia Chiasson, Alain Forget, Robert Biddle, and PC van Oorschot. Influencing users towards better passwords: persuasive cued click-points. *People and Computers XXII Culture, Creativity, Interaction 22*, pages 121–130, 2008.

[86] Alain Forget, Sonia Chiasson, Paul C Van Oorschot, and Robert Biddle. Improving text passwords through persuasion. In *Proceedings of the 4th symposium on Usable privacy and security*, pages 1–12, 2008.

[87] Daniel LeBlanc, Sonia Chiasson, Alain Forget, and Robert Biddle. Can eye gaze reveal graphical passwords? In *ACM Symposium on Usable Privacy and Security (SOUPS)*, 2008.

[88] Alain Forget, Sonia Chiasson, and Robert Biddle. Helping users create better passwords: Is this the right approach? In *Proceedings of the 3rd Symposium on Usable Privacy and Security*, pages 151–152, 2007.

[89] Rick Wash, Emilee Rader, Ruthie Berman, and Zac Wellmer. Understanding password choices: How frequently entered passwords are re-used across websites. In *Twelfth Symposium on Usable Privacy and Security (SOUPS)*, pages 175–188, Denver, CO, 2016. USENIX Association.

[90] Elizabeth Stobert and Robert Biddle. The password life cycle: User behaviour in managing passwords. In *10th Symposium On Usable Privacy and Security ((SOUPS) 2014)*, pages 243–255, Menlo Park, CA, USA, 2014. USENIX Association.

[91] Ameya Hanamsagar, Simon S Woo, Chris Kanich, and Jelena Mirkovic. Leveraging semantic transformation to investigate password habits and their causes. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, page 570, New York, NY, USA, 2018. ACM, ACM.

[92] Sarah Pearman, Shikun Aerin Zhang, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Why people (don't) use password managers effectively. In *Fifteenth Symposium On Usable Privacy and Security (SOUPS 2019). USENIX Association, Santa Clara, CA*, pages 319–338, 2019.

[93] Sarah Pearman, Jeremy Thomas, Pardis Emami Naeini, Hana Habib, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, Serge Egelman, and Alain Forget. Let's go in for a closer look: Observing passwords in their natural habitat. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 295–310, 2017.

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

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

[96] Luis Von Ahn, Manuel Blum, Nicholas J Hopper, and John Langford. Captcha: Using hard ai problems for security. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 294–311. Springer, 2003.

[97] Paul A Grassi, Elaine M Newton, Ray A Perlner, Andrew R Regenscheid, William E Burr, Justin P Richer, Naomi B Lefkovitz, Jamie M Danker, Yee-Yin Choong, Kristen Greene, et al. Digital identity guidelines: Authentication and lifecycle management. *Special Publication (NIST SP)-800-63B*, 2017.

[98] Joseph Bonneau and Sören Preibusch. The password thicket: Technical and market failures in human authentication on the web. In *WEIS*, 2010.

[99] Dinei Florencio and Cormac Herley. A large-scale study of web password habits. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 657–666, New York, NY, USA, 2007. ACM.

[100] Tom Le Bras. Online overload—it's worse than you thought. *Infographic, Dash Lane Blog, July*, 21, 2015.

[101] Daniel V Bailey, Markus Dürmuth, and Christof Paar. Statistics on password re-use and adaptive strength for financial accounts. In *International Conference on Security and Cryptography for Networks*, pages 218–235. Springer, 2014.

[102] Jim Finkle and Jennifer Saba. Linkedin suffers data breach-security experts. urlhttp://in.reuters.com/article/linkedin-breach-idINDEE8550EN20120606, May 18, 2017 visited.

[103] Poul-Henning Kamp, P Godefroid, M Levin, D Molnar, P McKenzie, R Stapleton-Gray, B Woodcock, and G Neville-Neil. Linkedin password leak: Salt their hide. *ACM Queue*, 10(6):20, 2012.

[104] Sarah Nadi, Stefan Krüger, Mira Mezini, and Eric Bodden. Jumping through hoops: Why do java developers struggle with cryptography apis? In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 935–946, New York, NY, USA, 2016. ACM.

[105] Yasemin Acar, Sascha Fahl, and Michelle L Mazurek. You are not your developer, either: A research agenda for usable security and privacy research beyond end users. In *Cybersecurity Development (SecDev), IEEE*, pages 3–8, Piscataway, NJ, USA, 2016. IEEE, IEEE Press.

[106] Mark A Schmuckler. What is ecological validity? a dimensional analysis. *Infancy*, 2(4):419–436, 2001.

[107] Martin Höst, Björn Regnell, and Claes Wohlin. Using students as subjects- a comparative study of students and professionals in lead-time impact assessment. *Empirical Software Engineering*, 5(3):201–214, 2000.

[108] Patrik Berander. Using students as subjects in requirements prioritization. In *Empirical Software Engineering, 2004. ISESE'04. Proceedings. 2004 International Symposium on*, pages 167–176, Redondo Beach, CA, USA, 2004. IEEE, IEEE.

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

[110] Mikael Svahnberg, Aybüke Aurum, and Claes Wohlin. Using students as subjects - an empirical evaluation. In *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '08, pages 288–290, New York, NY, USA, 2008. ACM.

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

[112] Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L Mazurek, and Christian Stransky. You get where you're looking for: The impact of information sources on code security. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 289–305, Piscataway, NJ, USA, May 2016. IEEE Press.

[113] Alena Naiakshina, Anastasia Danilova, Christian Tiefenau, Marco Herzog, Sergej Dechand, and Matthew Smith. 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, pages 311–328, New York, NY, USA, 2017. ACM.

[114] Spring. Website. Online: `https://spring.io/projects/spring-framework`; last accessed August 31, 2019.

[115] Javaserver faces (JSF). Website. Online: `https://javaee.github.io/javaserverfaces-spec/`; last accessed August 31, 2019.

[116] Alena Naiakshina, Anastasia Danilova, Christian Tiefenau, and Matthew Smith. Deception task design in developer password studies: Exploring a student sample. In *Fourteenth Symposium on Usable Privacy and Security ((SOUPS) 2018)*, pages 297–313, Baltimore, MD, USA, 2018. USENIX Association.

[117] Alena Naiakshina, Anastasia Danilova, Eva Gerlitz, Emanuel von Zezschwitz, and Matthew Smith. "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, pages 140:1–140:12, New York, NY, USA, 2019. ACM.

[118] Alena Naiakshina, Anastasia Danilova, Eva Gerlitz, and Matthew Smith. On conducting security developer studies with cs students: Examining a password-storage study with cs students, freelancers, and company developers. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, page 1–13, New York, NY, USA, 2020. Association for Computing Machinery.

[119] Matthew Finifter and David Wagner. Exploring the relationship betweenweb application development tools and security. In *Proceedings of the 2Nd USENIX Conference on Web Application Development*, WebApps'11, pages 9–9, Berkeley, CA, USA, 2011. USENIX Association.

[120] Githut: A small place to discover languages in github. `http://githut.info/`, February 6, 2018 visited.

[121] W3techs web technology surveys: 'usage of server-side programming languages for websites'. `https://w3techs.com/technologies/overview/programming_language/all`, February 6, 2018 visited.

[122] Tiobe index. `http://www.tiobe.com/tiobe-index/`, February 6, 2018 visited.

[123] Pypl popularity of programming language. `http://pypl.github.io/PYPL.html`, February 6, 2018 visited.

[124] The redmonk programming language rankings. `http://redmonk.com/sogrady/2017/06/08/language-rankings-6-17/`, February 6, 2018 visited.

[125] Trendy skills: Extracting skills that employers seek in the it industry. `http://trendyskills.com/`, February 6, 2018 visited.

[126] Hotframeworks: Web framework rankings. `http://hotframeworks.com/languages/java`, May 18, 2017 visited.

[127] Github: Built for developers. `https://github.com/`, May 18, 2017 visited.

[128] Stack overflow: Stack overflow is a community of 7.0 million programmers, just like you, helping each other. `https://stackoverflow.com/`, September 02, 2019 visited.

[129] Google trends: Visualizing google data. `https://trends.google.com/trends/`, May 18, 2017 visited.

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

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

[132] Allan J Kimmel, N Craig Smith, and Jill Gabrielle Klein. Ethical decision making and research deception in the behavioral sciences: An application of social contract theory. *Ethics & Behavior*, 21(3):222–251, 2011.

[133] Karl Spencer Lashley. *The problem of serial order in behavior*, volume 21. Bobbs-Merrill, 1951.

[134] John A Bargh. The historical origins of priming as the preparation of behavioral responses: Unconscious carryover and contextual influences of real-world importance. *Social Cognition*, 32(Supplement):209–224, 2014.

[135] John A Bargh and Tanya L Chartrand. The mind in the middle. *Handbook of research methods in social and personality psychology*, 2:253–285, 2000.

[136] Evan Weingarten, Qijia Chen, Maxwell McAdams, Jessica Yi, Justin Hepler, and Dolores Albarracin. From primed concepts to action: A meta-analysis of the behavioral effects of incidentally presented words. *Psychological bulletin*, 142 5:472–97, 2016.

[137] Stuart E Schechter, Rachna Dhamija, Andy Ozment, and Ian Fischer. The emperor's new security indicators. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 51–65. IEEE, 2007.

[138] P. G. Kelley, S. Komanduri, M. L. Mazurek, R. Shay, T. Vidas, L. Bauer, N. Christin, L. F. Cranor, and J. Lopez. Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. In *2012 IEEE Symposium on Security and Privacy*, pages 523–537, May 2012.

[139] Jason Bau, Frank Wang, Elie Bursztein, Patrick Mutchler, and John C Mitchell. Vulnerability factors in new web applications: Audit tools, developer selection & languages. *Stanford, Tech. Rep*, 2012.

[140] Rick Wash and Emilee Rader. Influencing mental models of security: a research agenda. In *Proceedings of the 2011 New Security Paradigms Workshop*, pages 57–66. ACM, 2011.

[141] Iulia Ion, Rob Reeder, and Sunny Consolvo. "... no one can hack my mind": Comparing expert and non-expert security practices. In *SOUPS*, volume 15, pages 1–20, 2015.

[142] Carol Sansone, Carolyn C Morf, and Abigaïl T Panter. *The Sage handbook of methods in social psychology*. Sage Publications, 2003.

[143] Allan J Kimmel. *Ethical issues in behavioral research: Basic and applied perspectives*. John Wiley & Sons, 2009.

[144] Allan J Kimmel and N Craig Smith. Deception in marketing research: Ethical, methodological, and disciplinary implications. *Psychology & Marketing*, 18(7):663–689, 2001.

[145] Hilarie Orman. Twelve random characters: Passwords in the era of massive parallelism. *IEEE Internet Computing*, 17(5):91–94, 2013.

[146] NIST FIPS PUB. 186,". *Digital Signature Standard," National Institute of Standards and Technology, US Department of Commerce*, 18, 1994.

[147] Donald E. Eastlake, 3rd and Paul E. Jones. Us secure hash algorithm 1 (sha1). RFC 3174 (Informational), September 2001.

[148] Ronald L. Rivest. The md5 message-digest algorithm. RFC 1321 (Informational), April 1992.

[149] Burt Kaliski. Pkcs# 5: Password-based cryptography specification version 2.0, September 2000.

[150] Niels Provos and David Mazieres. A future-adaptable password scheme. In *USENIX Annual Technical Conference, FREENIX Track*, pages 81–91, 1999.

[151] Colin Percival. Stronger key derivation via sequential memory-hard functions. *Self-published*, pages 1–16, 2009.

[152] John Kelsey, Bruce Schneier, Chris Hall, and David Wagner. Secure applications of low-entropy keys. In *International Workshop on Information Security*, pages 121–134. Springer, 1997.

[153] George Hatzivasilis, Ioannis Papaefstathiou, and Charalampos Manifavas. Password hashing competition-survey and benchmark. *IACR Cryptology ePrint Archive*, 2015:265, 2015.

[154] George Hatzivasilis, Ioannis Papaefstathiou, Charalampos Manifavas, and Ioannis Askoxylakis. Lightweight password hashing scheme for embedded systems. In *IFIP International Conference on Information Security Theory and Practice*, pages 260–270. Springer, 2015.

[155] Christian Forler, Stefan Lucks, and Jakob Wenzel. Catena: A memory-consuming password-scrambling framework. Technical report, Cryptology ePrint Archive, Report 2013/525, 2013.

[156] Christian Forler, Stefan Lucks, and Jakob Wenzel. Memory-demanding password scrambling. In *ASIACRYPT (2)*, pages 289–305, 2014.

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

[158] Password Hashing Competition and our recommendation for hashing passwords: Argon2, 2015.

[159] Christian Forler, Eik List, Stefan Lucks, and Jakob Wenzel. Overview of the candidates for the password hashing competition. In *International Conference on Passwords*, pages 3–18. Springer, 2014.

[160] Meltem Sönmez Turan, Elaine B Barker, William E Burr, and Lidong Chen. Sp 800-132. recommendation for password-based key derivation: Part 1: Storage applications. 2010.

[161] Garcia Michael E Fenton James L Grassi, Paul A. Draft nist special publication 800 63 3 digital identity guidelines. 2017.

[162] Ben Alex, Luke Taylor, Rob Winch, and Gunnar Hillert. Spring security reference: 4.1.3.release, 2004-2015.

[163] Friedrich Wiemer and Ralf Zimmermann. High-speed implementation of bcrypt password search using special-purpose hardware. In *ReConFigurable Computing and FPGAs (ReConFig), 2014 International Conference on*, pages 1–6. IEEE, 2014.

[164] Katja Malvoni and Josip Knezović. Are your passwords safe: Energy-efficient bcrypt cracking with low-cost parallel hardware. In *WOOT'14 8th Usenix Workshop on Offensive Technologies Proceedings 23rd USENIX Security Symposium*, 2014.

[165] Markus Dürmuth and Thorsten Kranz. On password guessing with gpus and fpgas. In *International Conference on Passwords*, pages 19–38. Springer, 2014.

[166] OWASP. Password storage cheat sheet. Website, 2014. Online `https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet`; last accessed May 18, 2017.

[167] Sascha Fahl, Marian Harbach, Thomas Muders, Lars Baumgärtner, Bernd Freisleben, and Matthew Smith. Why eve and mallory love android: An analysis of android ssl (in) security. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 50–61. ACM, 2012.

[168] Sascha Fahl, Marian Harbach, Marten Oltrogge, Thomas Muders, and Matthew Smith. Hey, you, get off of my clipboard. In *International Conference on Financial Cryptography and Data Security*, pages 144–161. Springer, 2013.

[169] Richard A Armstrong. When to use the Bonferroni correction. 34:502–508, 2014.

[170] Peter Mayer, Jan Kirchner, and Melanie Volkamer. 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)*, pages 13–28, Santa Clara, CA, 2017. USENIX Association.

[171] Rebecca Balebako, Abigail Marsh, Jialiu Lin, Jason I Hong, and Lorrie Faith Cranor. The privacy and security behaviors of smartphone app developers. 2014.

[172] Chamila Wijayarathna and Nalin A. G. Arachchilage. 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, pages 205–210, New York, NY, USA, 2018. ACM.

[173] Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher Kruegel. An empirical study of cryptographic misuse in android applications. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 73–84. ACM, 2013.

[174] David Lazar, Haogang Chen, Xi Wang, and Nickolai Zeldovich. Why does cryptographic software fail?: a case study and open problems. In *Proceedings of 5th Asia-Pacific Workshop on Systems*, page 7. ACM, 2014.

[175] PL Gorski and L Lo Iacono. Towards the usability evaluation of security apis. In *Proceedings of the Tenth International Symposium on Human Aspects of Information Security & Assurance (HAISA 2016)*, page 252. Lulu. com, 2016.

[176] Jeffrey Stylos and Brad A Myers. The implications of method placement on api learnability. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 105–112. ACM, 2008.

[177] Brad A Myers and Jeffrey Stylos. Improving api usability. *Communications of the ACM*, 59(6):62–69, 2016.

[178] Ulrich Stärk, Lutz Prechelt, and Ilija Jolevski. Plat_forms 2011: Finding emergent properties of web application development platforms. In *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 119–128. ACM, 2012.

[179] Lutz Prechelt and Ulrich Stärk. Plat_forms: Contests as an alternative approach to se empirical studies in industry. In *Proceedings of the 1st International Workshop on Conducting Empirical Studies in Industry*, pages 59–62. IEEE Press, 2013.

[180] J. Xie, H. R. Lipford, and B. Chu. Why do programmers make security errors? In *2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 161–164, Piscataway, NJ, USA, Sep. 2011. IEEE Press.

[181] Elissa M Redmiles, Amelia R Malone, and Michelle L Mazurek. I think they're trying to tell me something: Advice sources and selection for digital security. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 272–288. IEEE, 2016.

[182] Elissa M. Redmiles, Sean Kross, and Michelle L. Mazurek. How i learned to be secure: A census-representative survey of security advice sources and behavior. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 666–677, New York, NY, USA, 2016. ACM.

[183] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. Why don't software developers use static analysis tools to find bugs? In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 672–681, Piscataway, NJ, USA, 2013. IEEE Press.

[184] Tyler W. Thomas, Heather Lipford, Bill Chu, Justin Smith, and Emerson Murphy-Hill. What questions remain? an examination of how developers understand an interactive static analysis tool. In *Twelfth Symposium on Usable Privacy and Security ((SOUPS) 2016)*, Denver, CO, June 2016. USENIX Association.

[185] T. Thomas, B. Chu, H. Lipford, J. Smith, and E. Murphy-Hill. A study of interactive code annotation for access control vulnerabilities. In *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 73–77, Piscataway, NJ, USA, Oct 2015. IEEE Press.

[186] Felix Fischer, Konstantin Böttinger, Huang Xiao, Christian Stransky, Yasemin Acar, Michael Backes, and Sascha Fahl. Stack overflow considered harmful? *The Impact of Copy & Paste on Android Application Security. CoRR abs/1710.03135*, 2017.

[187] Shubham Jain and Janne Lindqvist. Should i protect you? understanding developers' behavior to privacy-preserving apis. In *Workshop on Usable Security*, volume 2014, 2014.

[188] Titus Barik, Justin Smith, Kevin Lubick, Elisabeth Holmes, Jing Feng, Emerson Murphy-Hill, and Chris Parnin. Do developers read compiler error messages? In *Proceedings of the 39th International Conference on Software Engineering*, ICSE '17, pages 575–585, Piscataway, NJ, USA, 2017. IEEE Press.

[189] L. Layman, L. Williams, and R. S. Amant. Toward reducing fault fix time: Understanding developer behavior for the design of automated fault detection tools. In *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, pages 176–185, Sep. 2007.

[190] R. Scandariato, J. Walden, and W. Joosen. Static analysis versus penetration testing: A controlled experiment. In *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*, pages 451–460, Piscataway, NJ, USA, Nov 2013. IEEE Press.

[191] Aiko Yamashita and Leon Moonen. 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*, pages 5–8, San Francisco, CA, USA, 2013. IEEE, IEEE.

[192] Aiko Yamashita and Leon Moonen. Do developers care about code smells? an exploratory survey. In *Reverse Engineering (WCRE), 2013 20th Working Conference on*, pages 242–251. IEEE, IEEE, 2013.

[193] Duc Cuong Nguyen, Dominik Wermke, Yasemin Acar, Michael Backes, Charles Weir, and Sascha Fahl. 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, pages 1065–1077, New York, NY, USA, 2017. ACM.

[194] Kraig Finstad. Response interpolation and scale sensitivity: Evidence against 5-point scales. *Journal of Usability Studies*, 5(3):104–110, 2010.

[195] William Albert and E Dixon. Is this what you expected? the use of expectation measures in usability testing. In *Proceedings of the Usability Professionals Association 2003 Conference, Scottsdale, AZ*, 2003.

[196] Donna Tedesco and Tom Tullis. A comparison of methods for eliciting post-task subjective ratings in usability testing. *Usability Professionals Association (UPA)*, 2006:1–9, 2006.

[197] Jeff Sauro and Joseph S Dumas. Comparison of three one-question, post-task usability questionnaires. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1599–1608. ACM, 2009.

[198] Jeff Sauro. If you could only ask one question, use this one. `http://www.measuringu.com/blog/single-question.php`, May 18, 2017 visited.

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

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

[201] I Standard. Ergonomic requirements for office work with visual display terminals (vdts)–part 11: Guidance on usability. iso standard 9241-11: 1998. *International Organization for Standardization*, 1998.

[202] S M Taiabul Haque, Matthew Wright, and Shannon Scielzo. A Study of User Password Strategy for Multiple Accounts. pages 1–3.

[203] Alain Forget, Sonia Chiasson, P C Van Oorschot, and Robert Biddle. Improving Text Passwords Through Persuasion. In *Proceedings of the 4th Symposium on Usable Privacy and Security*, pages 1–12. ACM, jul 2008.

[204] Roger E Kirk. *Experimental design*. Wiley Online Library, 1982.

[205] Glipper is a clipboardmanager for gnome. `https://launchpad.net/glipper`, February 6, 2018 visited.

[206] John Brooke et al. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.

[207] S Clarke. Using the cognitive dimensions framework to design usable apis.

[208] Jakob Nielsen. *Usability engineering*. Elsevier, 1994.

[209] Philip G. Inglesant and M. Angela Sasse. 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, pages 383–392, New York, NY, USA, 2010. ACM.

[210] Kamran Siddiqui. Heuristics for sample size determination in multivariate statistical techniques. *World Applied Sciences Journal*, 27:285–287, 01 2013.

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

[212] Awanthika Senarath and Nalin Asanka Gamagedara Arachchilage. Understanding software developers' approach towards implementing data minimization. *arXiv preprint arXiv:1808.01479*, 2018.

[213] Chamila Wijayarathna and Nalin AG Arachchilage. Am i responsible for end-user's security? Baltimore, MD. USENIX Association.

[214] Thomas D. LaToza, Gina Venolia, and Robert DeLine. Maintaining mental models: A study of developer work habits. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 492–501, New York, NY, USA, 2006. ACM.

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

[216] Open web application security project (OWASP). Accessed: September 2019.

[217] Dan Arias. Hashing in action: Understanding bcrypt, 2018.

[218] Dan Boneh, Henry Corrigan-Gibbs, and Stuart Schechter. Balloon hashing: A memory-hard function providing provable protection against sequential attacks. Cryptology ePrint Archive, Report 2016/027, 2016. https://eprint.iacr.org/2016/027.

[219] Xing. Website. Online: https://www.xing.com/; last accessed August 31, 2019.

[220] Linkedin. Website. Online: https://www.linkedin.com; last accessed September 2019.

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

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

[223] Random (java se 11 & jdk 11) - oracle docs. Website, 2014. Online: https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Random.html; last accessed September 05, 2019.

[224] Securerandom (java se 11 & jdk 11) - oracle docs. Website, 2014. Online: https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/security/SecureRandom.html; last accessed September 05, 2019.

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

[226] Bundesamt für sicherheit in der informationstechnik (bsi) - passwoerter. Website. Online: https://www.bsi-fuer-buerger.de/BSIFB/DE/Empfehlungen/Passwoerter/passwoerter_node.html;jsessionid=0556EAA71C3FCA1F0A9083485BD616B0.2_cid341; last accessed September 28, 2020.

[227] Stefan Krüger, Sarah Nadi, Michael Reif, Karim Ali, Mira Mezini, Eric Bodden, Florian Göpfert, Felix Günther, Christian Weinert, Daniel Demmler, et al. Cognicrypt: supporting developers in using cryptography. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 931–936. IEEE, 2017.

[228] Stefan Krüger, Karim Ali, and Eric Bodden. Cognicrypt gen: generating code for the secure usage of crypto apis. In *Proceedings of the 18th ACM/IEEE International Symposium on Code Generation and Optimization*, pages 185–198, 2020.

[229] Albrecht Schmidt. Don't blame the user: toward means for usable and practical authentication. *interactions*, 26(3):73–75, 2019.

[230] Visit Almaty. Pexels.com: Photo of three persons holding bubble jacket carrying snowboards. https://www.pexels.com/photo/three-person-holding-bubble-jacket-carrying-snowboards-848594/, January 02, 2019 visited.

[231] Alexandra. Pixabay.com: girl gate curious play out nature. https://pixabay.com/en/girl-gate-curious-play-out-nature-1424937/, January 02, 2019 visited.

[232] Teodor Andersson. Pixabay.com: Photo football boy youth teen run shoes. `https://pixabay.com/en/football-boy-youth-teen-run-shoes-2395754/`, January 02, 2019 visited.

[233] Keith Johnston. Pixabay.com: Photo american football. `https://pixabay.com/de/american-football-1465510/`, January 02, 2019 visited.

[234] Noz Urbina. Pexels.com: Photo of people snorkeling underwater. `https://www.pexels.com/photo/photo-of-people-snorkeling-underwater-734725/`, January 02, 2019 visited.