On the Importance of Ecologically Valid Usable Security Research for End Users and IT Workers

Der Fakultät für Elektrotechnik und Informatik
der Gottfried Wilhelm Leibniz Universität Hannover
zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften

Dr. rer. nat.

genehmigte Dissertation von

Dipl.-Inf. Sascha Fahl
geboren am 23.03.1985
in Eschwege

2016

Referent: Prof. Dr. Matthew Smith
Koreferent: Prof. Dr. Robert Jäschke
Tag der Promotion: 30.05.2016

Foreword

The research that is the heart of this thesis would not have been possible without the support of my advisor, co-authors, colleagues, students, research assistants, friends and my family.

I owe my deepest gratitude to my family. My family has always encouraged me to be curious, tenacious, and ambitious. My parents, Jutta and Albert Fahl, are the ones who made all this possible from the beginning, for which I will be eternally thankful. I am grateful to my wonderful wife Yasemin, who always provided me with her patience and support and helped me through all the long nightshifts we went through over the last three years. Your support was manifold: You helped me picking the right research questions, pushed me when I was unmotivated and made the best soy latte when we were working nightshifts. I dedicate this to you and our wonderful marriage.

I would like to thank my advisor Matthew Smith for being a great supervisor. He gave me the chance to work on almost all ideas I had as a doctoral candidate, encouraging me to tackle all the challenges working towards a doctor's degree keeps ready and made me a thoroughbred researcher.

I am also grateful to Professor Robert Jäschke, for being my secondary examiner. Gabriele von Voigt, head of the distributed systems institute, did a great job as the second-in-command and always was a sensitive and reliable discussion partner. Also, my colleagues at the Distributed Systems and Security Group, Henning Perl, Michael Brenner, Thomas Muders, Hinrich Tobaben, Benjamin Henne, Christian Szongott, and Jan Wiebelitz, made for a very pleasant and inspiring working environment. Furthermore, I would like to thank Sergej Dechand at University Bonn, with whom it was great to collaborate on multiple projects; Jaromir Smrcek from Zoner Inc., who provided me access to very interesting telemetry data of their Android Anti-Virus software; and finally Adrienne Porter-Felt from Google, who made parts of my Ph.d. student time as an intern at the Chrome Usable Security team in Mountain View a very special experience.

I enjoyed collaborating on several projects with the greatest research assistants I could imagine: Christian Stransky, Rafal Lesniak, Felix Fischer, Marten Oltrogge, Oliver Weidner, Pascal Urban, Steffen Busch, Maximilian Kaulmann, Richard Neumann, Jannis König, Mohamed Brahim Rekik and Roman Zimbelmann. Finally, I would like to thank the students that wrote their Bachelor and Master theses under my supervision: Marten Oltrogge, Christian Stransky, Markus Kötter, Felix

Fischer, Falk Garbsch, Pascal Urban, Eugen Bier, Jannis König, David Bluhm, Leif Erik Wagner, Toni Bartilla, Marcus Wobig and Andre Brinkop.

Hannover, June 19, 2016

Summary

In past research, the focus of IT security research was in investigating and developing novel IT security mechanisms and technologies such as cryptographic primitives and protocols. However, over the last decade, researchers in academia and industry realized that human factors are of crucial importance when developing and deploying IT security mechanisms. Since then, usable security and privacy research attracts increased interest, focusing on improvement of the usability of human interaction with IT security mechanisms of any kind.

Researchers for example investigated password based authentication or the comprehension and adherence of warning messages. The majority of previous works focused on end users of IT security measures and collects data in first contact studies, making intensive use of self-reporting questions.

However, one important question has not received adequate attention in the usable security and privacy research community:

What is the impact of and interdependence between IT security measures for end users, administrators, software developers and system designers on the overall security and usability of IT ecosystems?

In this thesis, I present and discuss the results of seven research projects with end users, administrators, developers and system designers.

First, I present a study on the usability and acceptance of a security measure to encrypt Facebook private messages. Second, I present and discuss multiple studies on the usability of programming and configuration interfaces for administrators and developers. Results illustrate that usability issues for administrators and developers working with IT security mechanisms have a huge impact on the security and usability of end user measures. Subsequently, I present a novel mechanism that allows easy and straightforward verification of the authenticity and integrity of software. I also discuss usability advantages for app market designers and end users.

While working on the above topics, I found that a large fraction of previous works that focused on user studies mention concerns about the ecological validity of their study designs and results. Those concerns address the question of whether data gathered in user studies is valid and allows to draw reliable conclusions. However, there is no dedicated work that addresses these questions. To motivate more foundational research in that area, I present and discuss a study on the ecological validity of a password study.

To the best of my knowledge, my work is the first to reveal the importance of usable security and privacy research to address human factors for end users, administrators, developers and system designers to improve overall usability and security. Moreover, I demonstrate the relevance of foundational research that concentrates on ecological validity of user study designs, data collection and result reporting for future usable security and privacy research.

Keywords: Usable Security, End Users, IT Workers

Zusammenfassung

Der Fokus von IT-Sicherheitsforschung lag lange Zeit vornehmlich in der Erforschung und Entwicklung neuer Sicherheitsmechanismen und -technologien wie zum Beispiel kryptografischen Verfahren und Protokollen. Erst in den letzten Jahren wurde Forschern in Wissenschaft und Industrie bewusst, dass der Mensch und seine Bedürfnisse eine wichtige Rolle bei der Entwicklung und vor allem der Verbreitung und Akzeptanz von Sicherheitsmaßnahmen spielen. Die seitdem stetig bedeutsamer werdende Forschung zu benutzbarer IT-Sicherheit und Privatsphäre beschäftigt sich mit der Verbesserung von Benutzbarkeit der Interaktion zwischen Benutzern und IT-Sicherheitsmechanismen jeglicher Art.

Intensiv erforschte Bereiche sind beispielsweise Passwörter zur Authentifizierung oder die Verständlichkeit und Effektivität von Warnungsmeldungen. Ein Großteil der vergangenen Forschungsarbeiten konzentriert sich auf Endnutzer von IT-Sicherheitsmechanismen und erhebt Daten im Rahmen von Erst-Kontakt Studien mit hohem Selbsteinschätzungsanteil. Eine zentrale Fragestellung wurde in der Usable Security and Privacy Forschungsgemeinde bisher jedoch relativ wenig beachtet:

Worin besteht die Bedeutung und die gegenseitige Abhängigkeit von IT-Sicherheitsmechanismen für Endnutzer, Administratoren, Softwareentwickler und Konstrukteure von IT-Systemen in Bezug auf die globale Sicherheit und Benutzbarkeit von IT-Ökosystemen?

Im Rahmen dieser Arbeit gehe ich daher auf Ergebnisse von sieben Forschungsarbeiten mit Endnutzern, Administratoren, Entwicklern und Konstrukteuren von IT-Systemen ein. Zunächst stelle ich eine Analyse der Benutzbarkeit und Akzeptanz von Sicherheitsmaßnahmen zur Verschlüsselung von privaten Nachrichten auf Facebook vor. Danach gehe ich auf verschiedene Studien zur einfachen Benutzbarkeit von Programmier- und Konfigurationsschnittstellen für Entwickler und Administratoren ein. Die Ergebnisse zeigen auf, dass Benutzbarkeitsprobleme auf Entwickler- und Administratorenebene maßgeblichen Einfluss auf die Sicherheit und Benutzbarkeit von Endanwendersystemen und -sicherheitsmechanismen haben. Im Anschluss stelle ich einen neuartigen Mechanismus zur einfachen Überprüfung der Authentizität und Integrität von Software vor und diskutiere Benutzbarkeitsvorteile für Konstrukteure von App-Markets und End-Nutzer.

Viele vorherige Arbeiten, die sich mit Benutzerstudien beschäftigt haben, sprechen die Problematik der ökologischen Validität von Studienergebnissen an. Im Kern dieser Fragestellung geht es darum, einzuschätzen, ob in Benutzerstudien erhobene Daten valide und verlässliche Aussagen erlauben. Es gibt jedoch keine Vorarbeiten, die sich gezielt mit diesen Fragen beschäftigen. Als Ausblick auf zukünftige zentrale Grundlagenforschung im Bereich in der Usable Security and Privacy Forschung stelle ich zuletzt eine Studie zur ökologischen Validität von Forschungsergebnissen über Passwortstudien vor.

Insgesamt macht die vorliegende Arbeit als eine der ersten deutlich, dass sich Forschungsarbeiten zu Usable Security and Privacy mit menschlichen Faktoren, die auf Endanwender, Administratoren, Entwickler, und Konstrukteure von IT-Systemen wirken, beschäftigen müssen. Weiterhin wird außerdem aufgezeigt, dass Grundlagenforschung zu Fragestellungen der ökologischen Validität von Benutzerstudien ein wichtiger Gegenstand zukünftiger Usable Security and Privacy Forschung

sein sollte.

Schlagwörter: Benutzbare IT-Sicherheit, Endanwender, IT Arbeiter

Contents

1 Introduction **13**
1.1 Contributions . . . 16
1.2 About this Thesis . . . 17

2 Background **19**
2.1 Usable Security and Privacy Research . . . 20
2.2 Research Methods . . . 21
2.3 Background and Related Work . . . 23
2.3.1 Passwords . . . 23
2.3.2 Email Encryption . . . 26
2.3.3 Transport Layer Security . . . 27
2.3.4 Appified Platforms . . . 29
2.4 Summary . . . 30

3 End Users: Encrypting Facebook Messages **32**
3.1 Motivation . . . 33
3.2 Background . . . 34
3.3 Exploratory Phase . . . 36
3.4 Usable Facebook Message Encryption . . . 45
3.5 Evaluation . . . 48
3.6 Limitations . . . 56
3.7 Summary . . . 56

4 Administrators: Configuring HTTPS Webservers **58**
4.1 Motivation . . . 59
4.2 Background . . . 60
4.3 Administrator Study . . . 61
4.4 Discussion . . . 69
4.5 Limitations . . . 71
4.6 Summary . . . 72

5 Developers: Implementing Password Managers **74**
5.1 Motivation . . . 75
5.2 Background . . . 76
5.3 Password Sniffing on Android . . . 79
5.4 Security Analysis . . . 81
5.5 Developer Study . . . 85
5.6 Countermeasures . . . 86

5.7 USecPassBoard User Interface . . . 88
5.8 Summary . . . 90

6 Developers: Customizing Certificate Validation 92
6.1 Motivation . . . 93
6.2 Background . . . 95
6.3 Evaluating Android TLS Usage . . . 97
6.4 Userstudy: TLS Warning Messages . . . 107
6.5 Summary . . . 109

7 System Designers: Rethinking TLS Development 111
7.1 Motivation . . . 112
7.2 Background . . . 113
7.3 TLS on iOS . . . 115
7.4 Developer Study . . . 118
7.5 TLS Development Re-thought . . . 123
7.6 Limitations . . . 136
7.7 Summary . . . 136

8 System Designers: Distributing Software in a Bullet-Proof Way 138
8.1 Motivation . . . 139
8.2 Background . . . 141
8.3 App Signing Practices . . . 144
8.4 Threat Model . . . 147
8.5 Application Transparency . . . 148
8.6 Summary . . . 160

9 Closing the Ivory Gap: Ecological Validity 162
9.1 Motivation . . . 163
9.2 Background . . . 164
9.3 A Study of Studying Passwords . . . 167
9.4 Results . . . 177
9.5 Limitations . . . 182
9.6 Summary . . . 184

10 Conclusions 187
10.1 Future Work . . . 190

A Appendix: Message Encryption Study 192
A.1 Questionnaire Items . . . 192
A.2 Interview Guideline . . . 195

B Appendix: Webmaster Study 197
B.1 Contact Email . . . 197

C Appendix: Studying Android's TLS Warning Message 198
C.1 Online Survey . . . 198

D Appendix: Ecological Validity of a Password Study 202
D.1 Question Plan 202
D.2 Contingency Tables 203

Bibliography 206

Curriculum Vitae 218

1 Introduction

Over the last decades, academic and industry research contrived novel concepts and tools to improve information security.

From a conceptual point of view, many of those mechanisms provide (almost) perfect security. Users' information can be efficiently protected against all different sorts of attacks and attackers. Email can be PGP [211] or SMIME [191] encrypted and signed, network security mechanisms such as virtual private networks (VPN) [89] and transport layer security (TLS) [52] can secure information in transit, encrypted filesystems and databases can protect persistently stored data and strong authentication mechanisms like client certificates [52] or security tokens can be used to protect information against unauthorized access.

Leaving the ivory tower and stepping into the world of average Internet users, administrators of IT systems, software developers and IT system designers, reality looks very different.

The Snowden revelations [103] show that at least powerful nation state attackers such as intelligence services attack users' data regularly and on a global scale. In the era of organized cybercrime, data breaches are commonplace [150]. Given a report by Facebook [72], even after the Snowden revelations, most Email is still not sent over encrypted channels, giving everyone with access to the network instant access to the Emails' content. Although no data for the use of end-to-end Email encryption such as PGP or S/MIME is available, their deployment might be even lower. Also, the deployment of HTTPS as the secure protocol in the web is not as advanced as it should be [132], allowing attackers with network access to easily snoop on and even modify data in transit. Researchers (e. g. [21, 48]) demonstrated that many users (re-)use weak passwords and service providers do not properly secure their users' data [150].

A large number of studies (e. g. [188, 114, 2, 130, 201, 82, 83]) investigates human factors of IT security measures. These studies identify underlying root causes responsible for hindering widespread adoption of IT security measures or stimulate their insecure usage.

However, not only careless or uneducated Internet users fail to make use of the many oftentimes effective but complicated security measures that are available today. Additionally and maybe more gravely, professional information workers such as system administrators, software developers and IT system designers contribute to the state of fragmentary information security today.

The current deployment of TLS is an overwhelming example: TLS warning messages are shown to browser users whenever a server's certificate could not be successfully validated. Those warning messages are intended to inform users of a serious attack. However, previous studies show that users oftentimes ignore or do not understand them (e. g. [82, 188]). Internet wide measurements (e. g. [3, 4, 73]) demonstrated that handling TLS certificates is not just challenging for end users, but also for administrators of webservers. They showed that a significant amount of TLS certificate configurations are misconfigured and trigger false warning messages. This leads to users seeing TLS warning messages in harmless cases, which contributes gravely to the problem. Users usually are overwhelmed by distinguishing the just mentioned harmless cases from erroneous certificates issued by valid certificate au-

thorities – either accidentally or deliberately – were used to attack real Internet users in the past [1]. In addition to end users and administrators of webservers, also software developers who deal with secure network connections contribute to the fragile condition of the TLS ecosystem: They regularly fail to implement non-default but secure certificate validation [76, 96] leaving their applications vulnerable to Man-In-The-Middle attacks.

This reality check impressively illustrates to what extent the current lack of usability in IT security measures – at all layers – limits the effectiveness of protecting information in the digital world.

The usable security and privacy research community has identified and investigated only a part of the problem: 15 years ago, Sasse et al. [2] were the first to investigate the fact that security is almost never the user's primary task. Most of the previous works stayed in the line of Sasse et al. and focused on end users. For example, Whitten and Tygar [201] conducted early and frequently cited work in our field, investigating the root causes of why users were unable to use PGP 5.0 correctly. Thereafter, the usable security and privacy community was established and in 2008, the first book on usable security and privacy research was released by Cranor and Garfinkel. From then on, the awareness of the importance of the usability of security and privacy mechanisms has increased and stimulated a positive development in our community. Warning messages today are easier to understand than a decade ago [188, 82], password meters may help users to create stronger passwords [193, 61], the comprehension and adherence of permission dialogs increased as demonstrated in studies over the last decade.

One common thread is that investigations of usability issues almost exclusively focus on end users. However, IT security measures also apply for other actors in IT ecosystems such as administrators, software developers and system designers. In contrast to end users, their impact on overall usability and security is much stronger and has yet not been the focus of extensive research.

Taken the above reality check and the development of our community within the last decade into account, it is obvious that studying end users alone is not enough. Instead, I argue that usable security and privacy research should focus on multiple actors in an IT (security) ecosystem:

End Users are the main audience of IT security measures and were the focus of most of the usable security and privacy research in the past. When used in real world scenarios such as authentication in the web or warning messages for TLS, IT security measures need to prove their effectiveness. Since end users' primary tasks usually do not focus on security, effectiveness means both secure and usable. If not usable, end users regularly circumvent IT security measures. Instead of perfect security in theory, their data stays unprotected in reality.

Administrators and Developers are important users of IT security systems and mechanisms: They are responsible for the secure development of software and the administration and configuration of computer systems. IT security

[1] cf. `https://blog.comodo.com/other/the-recent-ra-compromise/` – last access 13.04.2016

measures they employ, should be designed and implemented with information workers' needs in mind: Administrators and developers oftentimes do not have a strong IT security background and their primary tasks include the implementation of new features or the operation of production systems. Being of secondary or tertiary priority, security measures need to be easily implementable and configurable.

System designers decide about the deployment and implementation of IT security measures in their products and should balance between their economic interests and their users' security and privacy needs. Since their focus usually is monetization, complicated security mechanisms that scare their users are out of the question. Instead, security measures must not narrow their features and should be as easy to use as possible. Ideally, system designers want an IT security measure to provide additional value for their services.

Addressing all the above actors leverages the understanding of essential problems in current IT systems regarding users' information security and privacy. A better understanding will help to more effectively protect sensitive user information.

1.1 Contributions

In this thesis, I extend the state of the art of human factors in IT security measures: I present multiple case studies of the human factors in different areas of IT security dealing with end users, administrators, developers and system designers. Moreover, I demonstrate the importance of a holistic approach when working on usable security and privacy topics: If and only if the most important actors in IT ecosystems are included, IT security measures can achieve best efficiency.

I begin with illustrating how human factors influence the deployment of IT security measures for end users. This demonstrates how end users reason about a message encryption mechanism and then decide for or against such mechanisms. This line of work provides novel insights into which factors play a central role when IT security measures for end users are deployed and which pitfalls should be avoided.

As a next step, I conduct multiple technical analyses and user studies with a focus on administrators and developers. Based on the insights from related work and the first chapter, I move on to the creators and maintainers of IT security measures.

I investigate usability issues administrators encounter when configuring HTTPS enabled webservers. I conduct a survey with administrators that use misconfigured TLS certificates. My research provides new insights into the challenges for administrators and root causes of misconfigured TLS certificates. Based on these findings, I suggest changes to workflows and tools that help typical – non IT security experts – administrators in their daily work.

I identify usability challenges for mobile app developers when confronted with security critical tasks. Based on interviews with developers, I extract common challenging patterns. The lesson to be learned from this line of my research is that most developers – very similar to end users and administrators – first are no IT security

experts and second do mostly perceive IT critical aspects of their work as superfluous obstacles in conflict with the timely fulfillment of their original tasks. Hence, it is crucial to offer them easy to use mechanisms, tools and APIs to allow a smooth integration of secure solutions.

I identify usability issues of TLS APIs in mobile app development, investigate their root causes and shed light on why many developers fail when implementing security related code. Using these findings as a foundation, I re-think how developers should interact with TLS code. I provide insights in how system designers should address security related challenges for developers working with their systems.

Afterwards, I present and evaluate a novel framework that can help designers of central software distribution ecosystems such as app markets to offer their users an easy to use mechanism to verify the authenticity and integrity of software. This framework secures the current central software distribution points that currently allow powerful attackers to easily deploy customized (malicious) software to users. My framework is an example for the importance of easy to use mechanisms system designers may deploy to increase their users' information security and privacy.

While working on different user studies and reviewing research focuses on usable security and privacy questions, one important aspect researchers in our community are confronted with again and again is the question of the ecological validity of their research methods and study designs. However, the community still lacks sufficient work that addresses these questions for usable security and privacy research. To motivate more extensive future work in this area and to provide an example of how to conduct a study that focuses on ecological validity, I question the design of a popular type of study: Previous works often investigates human aspects of the deployment of passwords as the most widely used authentication mechanism in the wild. Those studies usually let users create passwords in laboratory or online studies and then use them to draw conclusions about aspects such as password strengths, memorability and usability. My work provides new insights into the ecological validity of this study design. I also show in which way data collected in these studies can or cannot be used to draw generic conclusions.

In summary, my research shows that usable security and privacy research should follow a holistic approach to study end users, administrators, developers and IT system designers with the aim to improve overall usability of security and privacy mechanisms. To the best of knowledge my work is the first to demonstrate the importance of usable security and privacy research for end users, administrators, developers and system designers. Additionally, I motivate more extensive foundational future usable security and privacy research with a focus on ecologically valid research methods.

1.2 About this Thesis

My thesis follows both a constructive and analytical approach. I analyze weak points in IT security measures for end users, administrators, developers and system designers and based on these findings construct novel usable tools and methods. I chose

this hybrid approach as building easy to use and secure IT security measures needs to focus on all involved actors: System designers need to choose to use secure technologies and their decisions directly impact all remaining actors in the ecosystem. Developers and administrators need to be able to develop and configure computer systems. Their choices and limitations they need to work with are based on decisions of system designers and directly impact the security and usability of end users working with their software. End users interact with IT security measures provided by system designers, developers and administrators. Their interactions with IT security measures is mostly limited to their own information and does rarely affect other actors in IT ecosystems. While they can contribute to their indirect security in some ways, insecure choices by the other actors often affect them directly and can seldomly be rectified at the end user level.

I hope my work will prove to be an in-depth foundation for other researchers, system designers, developers and administrators interested in making security measures easy to use.

In the time of writing this thesis, I have published several peer reviewed high quality conference research papers as first author or co-author with the invaluable help of many colleagues.

In these research papers, I do not focus on one topic exclusively; instead I took care to investigate the important key areas that lack proper coordination (often due to unclear interfaces and usability problems) and spawn security issues.

For this thesis, I discuss the findings presented in seven of these research papers, that clearly show a central point: We – as a researchers' community – need to understand that improving security measures to protect information in transit or persistently stored in computer systems requires a holistic approach. We need to apply valid methods to obtain reliable results and need to include end users, administrators, developers and system designers of IT security measures.

Most chapters in this thesis are based on previously published peer reviewed research papers and are thus marked with a disclaimer that explains my personal as well as my co-authors' contribution to each of those works. I am deeply grateful to my co-authors for helping me to work on these ideas. Without them, a large part of my research would not have been possible.

2 Background

This chapter gives background information for the subsequent chapters. Both, usable security and privacy research methods as well as technical foundations needed to understand the remainder of this thesis are described. This chapter can be either read before proceeding to read the following chapters, or be consulted as needed for the chapters, which each target different research areas and employ different research methods.

2.1 Usable Security and Privacy Research

This section is intended to give a brief introduction in usable security and privacy research and gives some information on the methods that are commonly used by researchers and practitioners. With their groundbreaking work "Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0" in 1999 Whitten and Tyga [201] demonstrated the importance of usability and human computer interaction (HCI) as part of the design and implementation of IT security measures. Since then, researchers investigated usability aspects of security and privacy mechanisms extensively and published their results at top tier conferences such as IEEE Security & Privacy, ACM Computer Communications and Security (CCS), Networked Distributed Systems Security NDSS, USENIX Security, ACM SIGCHI or USENIX SOUPS. This movement accounts for the fact that studying usability and human factors had long before been recognized as an integral part of computer system design and implementation in general.

The security community tenaciously worked on creating mechanism that users can interact with easily and effortlessly. Different norms and standards express the importance of this topic. Usability engineer Jakob Nielsen for example describes five different components to carefully consider when designing and implementing a computer system [1]:

Learnability How easy is it for novice users to interact with a system for the first time?

Efficiency Having learned how to interact, how quickly can tasks be performed?

Memorability How easy is it to interact with the system after periods of non-use?

Errors How many errors occur during interaction? How does the system deal with errors? How hard is it for users to recover from errors?

Satisfaction How pleasant do users find the interaction to be?

Using usability principles helps to build IT systems in general and IT security systems in particular that humans can interact with easily and effortlessly. However, one should consider carefully that the usability of IT systems depends on which type of user is trying to achieve which goals in which context. Types of users range from

[1] `https://www.nngroup.com/articles/usability-101-introduction-to-usability/` – last access 07.02.2016

end users, administrators, software developers to system designers – i. e. all types of consumers and producers of IT security systems and mechanisms.

Usability research covers many different topics and aspects of IT security but focuses mainly on end users. Previous works evaluated how users handle passwords (e. g. [50, 61, 186, 114, 130, 23, 48, 21, 130]), apply encryption mechanisms (e. g. [192, 201, 92, 133, 179, 169]), interact with permission systems (e. g. [159, 83, 84, 161, 109, 123] and warning messages(e. g. [188, 184, 82]) and perceive threats and risks in the online world (e. g. [25, 27, 26, 108]).

2.2 Research Methods

Usable security and privacy research supplements other fields of IT security research with the research methods applied. For example, system security researchers apply formal methods to construct secure operating systems, malware analysts use reverse engineering and static and dynamic code analysis [65] and cryptographers apply formal security analyses and mathematical proofs [152] to demonstrate properties of their systems. In addition, usable security and privacy research applies empirical methods as found in sociology, psychology and the HCI community [137] with the aim to make existing IT security measures easier to use or to design usable IT security measures from the ground up. Researchers often make use of surveys, interviews, focus groups and laboratory and field studies to shed light onto new research questions. This section will give a very brief overview of research methods typically applied in usable security and privacy studies. A more in-depth introduction into the methods applied by usable security and privacy researchers can be found in [137]. To conduct usable security and privacy studies the ecological validity of a study is of outstanding importance: The ecological validity of a (user) study means that the methods, materials and setup of the study must approximate the real-world that is being examined [91].

Interviews typically are used to collect information from an individual at a time. The interviewee uses an interviewer guideline that can be open (i. e. only consisting of some important aspects), semi-structured (i. e, consisting some questions but leave enough room to flexibly react to the interviewee if necessary) or structured (i. e, consisting of a fixed list of questions, leaving no room for variations). Interviews are used to collect qualitative data and help to explore new fields [115].

Focus groups are similar to interviews: They are usually used to collect qualitative data. However, instead of interviewing one single interviewee, a focus group consists of multiple participants (usually six to ten people) and the researcher acts as a moderator, whose primary task is leading a discussion between the focus group members. Taking notes or recording a focus group may help researchers to explore new ideas and concepts the focus group members brought up during their discussion [115].

Surveys are a method to collect quantitative data [137]. They are used to make statistical inferences about the population that is surveyed. Surveys consist of sets of different types of questions (e. g. likert scales, yes/no, multiple choice or open questions) that heavily rely on self-reporting. Surveys can be conducted using pen

and paper or online. Online surveys' popularity increased over the last years due to the small effort required to reach out to a large population. Statistical methods [85] typically are used to infer general conclusions.

In a laboratory study, participants are invited to the researchers' facilities to test new technologies or mechanisms. Participants work on specific tasks and afterwards are interviewed or fill out surveys [137]. Laboratory studies can be used to collect qualitative or quantitative data and are typically used to test novel technologies or mechanisms.

In contrast to inviting participants to the researchers' facilities, in field studies, participants test technologies or mechanisms in their familiar environment (e. g. at home or at work) [137]. Similar to laboratory studies, participants commonly are interviewed or asked to fill out surveys.

For all types of studies, the participant selection process is important. Participants should be a representative sample of the population that is studied. However, the selection process is complex and requires special attention by the researcher [137]. For example, it is common practice to recruit university students as participants in (end user) studies. However, it is important not to ask students who are in some way (e. g. through teaching) affiliated with the researcher. Those participants are very likely biased and would falsify the study results. Recruiting professional information workers is more challenging since an easy to use recruiting platform as available for end users is missing. Researchers often circumvent this limitation by recruiting computer science students, posting studies at websites such as craigslist [2] or asking developers who published apps in Google Play (e. g. [155]).

While traditionally small surveys, interviews, focus groups and lab studies were the dominant methods found in research papers, a current trend is the use of crowdsourcing services such as Amazon Mechanical Turk [32]. MTurk knows two types of users: Requesters can create "human intelligence tasks" (HITs) which were originally introduced to help requesters with tasks computers cannot easily solve such as tagging or categorizing images or texts. Workers (MTurkers) can then choose those HITs, solve them and get payed a small amount of money. MTurk enjoyed great popularity in the HCI community in general and the usable security and privacy community in particular - it is used for simple online experiments or surveys. Although the population of MTurk workers is different from the general population and workers might be motivated to cheat to finish as many tasks as quickly as possible, MTurk enables researchers to have easy access to a large pool of potential participants, which is a major improvement compared to the past [32].

Another trend in usable security and privacy studies is to collect participant information via self-reporting questions (e. g. [163, 5, 7, 9, 35, 37, 56, 106, 126, 165, 121, 125, 172]). Those studies show participants tasks or questions and ask them to fulfil or answer them to the best of their knowledge and belief. Psychologists and sociologists report limitations of that methodology [55]. However, usable security and privacy researchers use it a lot and many results are based on subjective opinions of study participants.

Since ecological validity has been a concern for research projects (e. g. [124, 130,

[2]cf. `http://www.craigslist.com` – last access 13.04.2016

178, 118]), I investigate a fundamental question of usable security and privacy research with the aim to motivate more foundational research in the future: Which research methods and study designs produce ecologically valid results? While it is relatively simple and straightforward to create an online survey and ask users for their opinion, past experiences or future plans, drawing conclusions based on purely self-reported information might produce wrong and/or mis-leading results [55]. Therefore, it is important to evaluate the selection of methods and study design and compare study results with real world data as often and in as much detail as possible.

2.3 Background and Related Work

In this section, I give a brief introduction into the technical concepts required to understand the work in the remainder of this thesis. Additionally, this section introduces relevant related work. This section can be skipped and subsections can be referenced individually as needed to understand the work presented in the individual chapters.

2.3.1 Passwords

Text-based passwords are the most common, widespread and possibly the most debated authentication mechanism in use. Logins to personal computers, unlock solutions for mobile devices and online service accounts are the most prominent use cases for text-based passwords. The inherent conflict of creating usable (e.g. user memorable) but secure (e.g. hard to guess) passwords has kept security researchers busy ever since the introduction of passwords to computer systems in the 1960s. Bonneau et al. [23] provide an excellent overview of the challenges of finding a good replacement for passwords. Using passwords as the primary authentication mechanism has many drawbacks.

It is known that users tend to use insecure passwords that are often easy to compromise by attackers [17, 21, 50, 87, 141, 149].

Another problem with the widespread deployment of passwords is their re-use [48, 141]. Users register with many services on the Internet, each requiring them to create a new username/password tuple. Previous research has shown that users often deal with this password overload by re-using the same or similar passwords for multiple accounts [95, 178]. Password re-use is a serious concern, since one compromised password can make multiple user accounts vulnerable.

However, not only end users are careless with their passwords: Operators of online services also often fail to properly protect their users' passwords as demonstrated by several password leaks in the past [3].

Multiple excellent surveys and systematizations [22, 23, 38] provide an extensive overview of the field.

[3] http://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/ – last access 07.02.2016

A substantial amount of password and password policy advice is based on anecdotal evidence and theoretical security measures. However, particularly the last few years have seen an increasing number of academic studies into password security and usability. Previous works can be split into the following categories:

- Analyzing the security of real world passwords (leaks).
- Evaluating aspects such as memorability and security of passwords created in user studies.
- Investigating the usability and security of improvements of text based passwords.
- Proposing and evaluating alternatives to text based passwords.

Collecting leaked real world passwords and analyzing their security and crackability has attracted many researchers in the past [120, 21, 149, 131]. Bonneau [21] analyzed the guessability of an anonymized corpus of 70 million Yahoo! users' passwords. They estimate that text passwords provide less than 10 bits of security against online and 20 bits of security against offline attacks. Das et al. [48] study several hundred thousand leaked passwords from eleven web sites and investigate how an attacker can leverage a known password from one site to more easily guess that user's password for other sites. They report that 50% of their users re-used the same passwords across multiple accounts which allowed them to guess 30% of the passwords in their dataset in less than 100 attempts.

With the rise of usable security and privacy research and the persistence of passwords, the number of research papers that investigated usability aspects of passwords in user studies increased [149, 194, 88, 114, 177, 129, 186].

Inglesant and Sasse [114] conducted a qualitative study with 32 staff members who kept a password diary for one week. This way they collected 196 passwords which allowed them to investigate the effectiveness of the password policies in place. In follow up interviews, the researchers found that users are in general concerned to maintain security, but that current security policies are too hard to satisfy. They conclude that instead of focusing password policies on maximizing strength and frequency alone, policies also should be designed using human factors principles.

Ur et al. [194] used previous studies that documented features that lead to easily guessable passwords as an opportunity to ask users why they craft weak passwords. To understand the participants' motivation, Ur et al. conducted a qualitative interview study with 49 participants. Most of the participants had a well-defined process for creating passwords. Common strategies are adding "!" to the end of a password or using words that are difficult to spell to seemingly make passwords secure.

Stobert and Biddle [186] conducted a series of interviews to investigate how users cope with needing to keep track of a large number of accounts and passwords. While they found that most of their participants reused passwords and wrote them down, they identified diverse personalized strategies. Although many users obviously disregard security advice, they involve a fine-grained self-management of sensitive resources. The researchers identify a password life-cycle that illustrates the users' strategies limitations.

In addition to investigations of the weaknesses of text-based passwords, multiple mechanisms have been proposed to support the user in choosing more secure passwords (e.g. [176, 193, 129]). Most of them focus on encouraging stronger passwords using password meters.

Ur et al. [193] presented a 2,931-subject study of password creation in the presence of 14 different password meters, analysed their effects and proposed guidelines for the design of effective password meters. They found that password meters with a variety of visual security indications led users to choose longer passwords. However, to make passwords not only longer but also more resistant to cracking, meters that scored passwords stringently were most effective.

Komanduri et al. [129] investigated a mechanism called Telepathwords. Telepathwords is intended to discourage the creation of weak, predictable passwords. During password creation, it makes realtime predictions for the next character of a password the user will type. In an MTurk study with 2,560 participants, they found that participants using Telepathwords created significantly fewer weak passwords compared to conventional password creation policies.

Another proposal to reduce problems related to text passwords is to use password managers. These typically require that users remember only a master password. Password managers store ideally randomly generated account passwords and send them on behalf of the user. However, password managers have their own usability [41, 122, 168] and security [187, 181, 138] issues (cf. Chapter 5, Page 75).

Since text-based passwords have security and usability drawbacks and efforts to improve the current situation have not lead to significant improvements, researchers work on alternative mechanisms (e.g. [190, 189, 16]).

Graphical passwords use series of images instead of text as a secret to authenticate. Biddle et al.[16] investigated both security and usability of graphical passwords conducting a large literature review.

Thorpe et al. [190] introduced GeoPass, a geographic location-password scheme. GeoPass users choose one or multiple places as their password. To investigate its security, usability and memorability, the researchers conducted a multi-session in-lab/at-home study. They found that 97% of their participants could remember their location-password for 8-9 days. Their security analysis finds that GeoPass passwords have reasonable security against online guessing attacks – even considering social engineering.

Czeskis et al. introduced PhoneAuth [46] which uses a smartphone as an authentication token and intends to replace text-based passwords. They address usability and deployability troubling challenges conventional two-factor authentication schemes. PhoneAuth provides security assurances comparable to or stronger than conventional two-factor authentication mechanisms while offering the same authentication experience as text-passwords.

While the above selection is not meant to be complete, it gives a good overview of password (related) research in the last decade. This section provides helpful background for Chapters 3, 5, 9, which, in some form or another touch passwords and their usability and security challenges.

2.3.2 Email Encryption

Mechanisms for email encryption are a hotly debated topic in the security community. On the one hand technologies such as PGP [211] or S/MIME [191] have the potential to protect email content against eavesdroppers. On the other hand, widespread adoption is still missing. PGP and S/MIME both apply asymmetric cryptography and digital certificates to encrypt and sign emails. Since my thesis focuses on usability issues, no detailed overview of the different properties of secure encryption protocols – such as PGP and S/MIME – is given at this point. However, joint work with Unger et al. [192] provides a comprehensive systematization on protocols and mechanisms that can be used to implement secure messaging systems.

Traditionally, usable security and privacy researchers worked on helping users to encrypt their email. Therefore, this section will briefly describe previous research in this field.

In 1999, Whitten and Tygar's Johnny study [201] raised awareness of the usability problems in email encryption with PGP 5. Only one third of their twelve participants was able to send encrypted and signed emails in their 90-minute test. 25% of the participants accidentally sent confidential information without encryption. Whitten and Tygar found significant problems with the user interface and questioned PGP's analogy between cryptographic and physical keys. They concluded that the interface *"does not come even reasonably close to achieving our usability standard"* and that it *"does not make [exchanging secure email] manageable for average computer users"*.

Garfinkel and Miller [92] built and evaluated a system based on key continuity management (KCM). Their prototype, CoPilot, addressed the problem of finding other users' public keys by automatically extracting senders' keys from incoming messages. Their study revealed that after using CoPilot for less than an hour, users generally understood the advantages of securing their emails. They found that while the KCM approach generally improved security, only a third of the participants chose encryption for confidential data and most sent information in an unprotected fashion. Some participants expected their email program to protect them from making mistakes and said that if encryption was important, a system administrator would have configured the email program to send only encrypted messages. This is a strong indicator that message encryption systems need to provide clear information about the security of the outgoing messages and apply security mechanisms automatically whenever possible [133].

Sheng et al. [179] conducted a follow-up pilot study to Whitten and Tygar's Johnny study with six novice users in order to understand the usability of PGP 9 and Outlook Express 6.0. Compared to the prior study of PGP 5, Sheng et al. found that PGP 9 made improvements in automatically encrypting emails, but the key verification process was still problematic and signatures in PGP 9 were actually more problematic than in PGP 5.

Ruoti et al. [169] conducted a usability study of a privacy enhancing webmail system. Their system hid as many security related details as possible – i. e. it includes automatic key management and automatic encryption. They evaluated the impact of hidden and transparent security mechanisms on the usability of their system. They found that a significant number of their participants mistakenly sent

out unencrypted messages and did not trust the system at all. A follow-up study that introduced some manual steps – i. e. copy-and-pasting encrypted ciphertext – did not negatively impact the acceptance, but prevented users from mistakenly sending unencrypted messages and drastically increased the users' trust.

While the above selection is not meant to be complete, it gives a good overview of what email encryption (related) research looked like in the last couple of years. This section provides helpful background for Chapter 3, Page 33.

2.3.3 Transport Layer Security

Transport Layer Security (TLS) is a cryptographic protocol that was introduced to protect network communication from eavesdropping and tampering [47]. TLS is the successor of SSL [90] and was introduced to fix security issues of the SSL protocol. For brevity, we will refer to both protocols as TLS, since features and issues described in this thesis affect both SSL and TLS in the same way. It is the transport layer protocol used to secure plain HTTP connection. The TLS secured HTTP protocol is called HTTPS [166, 86]. To establish a secure connection, a client must securely gain access to the public key of the server. In most client/server setups, the server obtains an X.509 certificate that contains the server's public key and is signed by a Certificate Authority (CA). When the client connects to the server, the certificate is transferred to the client. The client must then validate the certificate.

The basic validation checks include: a) does the subject (CN) of the certificate match the destination selected by the client?; b) is the signing CA a trusted CA?; c) is the signature correct?; and d) is the certificate valid in terms of its time of expiry? Additionally, revocation of a certificate and its corresponding certificate chain should be checked, but downloading Certificate Revocation Lists (CRLs) or using the Online Certificate Status Protocol (OCSP [70]) is often omitted.

TLS does not come with a mechanism for the client to communicate the hostname of the server to which it intends to establish a secured connection to an TLS-enabled server. However, especially in the presence of multiple *virtual* hosts running at a single underlying IP address, it may be desirable for clients to provide this information. In order to communicate this server name information, an TLS extension – called Server Name Indication (SNI) – exists, wich allows a client to send the server name in the *client hello* message of an TLS handshake[19]. The server receiving the client hello message can then choose the appropriate X.509 certificate to match the requested SNI value and include this certificate in its *server hello* response.

In a Man-in-The-Middle attack (MITM attack), the attacker is in a position to intercept messages sent between communication partners. In a passive MITM attack, the attacker can only eavesdrop on the communication (attacker label: Eve), and in an active MITM attack, the attacker can also tamper with the communication (attacker label: Mallory). MITM attacks against mobile devices are somewhat easier to execute than against traditional desktop computers, since the use of mobile devices frequently occurs in changing and untrusted environments. Specifically, the use of open access points [116] and the evil twin attack [182] make MITM attacks against mobile devices a serious threat.

A particular version of a MITM attack is TLS stripping [142, 143] that downgrades HTTPS links on websites to HTTP links. A MITM attacker who captures a plain HTTP website is able to replace HTTPS links with HTTP links and hence weakens the security of end users and threatens their privacy. To prevent TLS stripping is important to use HTTPS exclusively and not mix HTTP with HTTPS connections. Mechanisms such as HSTS [112] and Public Key Pinning [71] extend the HTTPS protocol to effectively protect against TLS stripping.

TLS is fundamentally capable of preventing both Eve and Mallory from executing their attacks. However, the cases described above open up attack vectors for both Eve and Mallory. Trivially, the mixed-mode/no TLS case allows Eve to eavesdrop on non-protected communication.

In case X.509 certificate validation fails, modern browsers show their users warning messages. These warning messages might imply that a MITM attack is occurring. Another reason for failures are misconfigured servers, e. g. a webmaster did not update an expired certificate or operates a certificate for an invalid hostname.

If there is a chance that the warning message is a false positive (i. e. the website's administrator (deliberately) misconfigured the X.509 certificate), browsers will show a bypassable warning message, discouraging users from clicking through.

Warning messages illustrate that the TLS ecosystem poses usability challenges to two types of users: On the one hand correctly configuring TLS certificates is a challenge for webmasters, on the other hand deciding whether to proceed or not when confronted with a warning message is a challenge for end users.

TLS has attracted various research that investigates different aspects of TLS security and usability [44, 3, 57, 188, 4, 82, 96]. Previous research can be split into the following categories:

- Analyzing the security of the TLS protocol and its implementations.
- Scanning the Internet and investigating the security and usability of real world TLS deployments.
- Conducting user studies to investigate usability challenges of TLS (warning messages).

Since my thesis focuses on usability issues, no complete overview of previous research on TLS will be given here. In particular, this section does not cover resesarch papers that investigate TLS protocol and implementation flaws (e. g. [36, 30, 15]). However, Clark and van Oorschot [44] provide a comprehensive overview of previous work on security and usability challenges of TLS.

Akhawe et al. [3] passively collected TLS handshakes of multiple US universities and 300,000 users over a period of nine months in 2012 and 2013, concentrating on the frequency of X.509 certificate validation errors in TLS handshakes. Overall, they found that 98.46% of the 3.9 billion TLS handshakes they monitored validated correctly, while 1.54% failed for different reasons: 70.51% used an unknown issuer, 2.99% a self-signed certificate, in 7.65% of all handshakes the certificate was expired and 18.82% of all handshakes generated hostname validation errors. Due to the

unlikeliness of an actual Man-In-The-Middle attack, they assume all validation errors to be false positives. Durumeric et al. [57] presented ZMap – a fast internet-wide scanner – and conducted 110 scans of the world-wide HTTPS infrastructure over one year, collecting more than 42 million unique certificates of which 6.9 million were browser trusted. These numbers demonstrate that TLS certificate configuration is not a straightforward task.

In 2009, Sunshine et al. [188] conducted lab studies with over 400 internet users to evaluate the effectiveness of browser TLS warning messages as well as their human understandability, finding that participants made unsafe choices when confronted with warning messages. They suggest reducing the number of warning messages altogether, taking the decision whether to trust an unsafe connection or not out of the users' hands.

In 2013, Akhawe and Porter Felt [4] used Firefox and Chrome's telemetry feature to measure click-through rates for TLS (and other) warning messages for different browsers in situ. Over a period of two months, they collected 16,704,666 TLS warning impressions for Chrome and 10,976 for Firefox. However, they were not able to see the respective handshakes or certificates that led to the warnings, thus they assume that almost all warning messages they saw were false positives. These results leave room for improving TLS warning messages.

In 2015, Felt et al. [82] report on the task to design TLS warnings with the goal of improving comprehension and adherence for users. They conducted multiple user-studies using microsurveys and a field experiment with Chrome's telemetry feature. They report that they failed at their goal of designing an easy-to-understand warning message, although 30% of their participants chose to remain safe after seeing their warning. They attribute this fact to the opinionated design approach they chose that made it attractive to users but failed to improve comprehension.

Georgiev et al. [96] investigated TLS certificate validation implementations in the wild. They find that many certificate validation implementations are completely broken and thus vulnerable to MITM attacks. Applications and libraries at all levels are affected. This work arose in parallel to my research in Chapter 6, Page 93. Both, my work and the work done by Georgiev et al. inspired researchers to investigate in the field of Chapter 6, Page 93 (e. g. [110, 185, 13]).

2.3.4 Appified Platforms

Appified platforms have recently seen a boom: Android and iOS became the defacto standards for appified operating systems with hundreds of millions of users and millions of apps. In particular Android security and privacy research attracted various researchers over the last years, resulting in manifold research papers [10, 42, 43, 66, 83, 84, 117, 127, 147, 156, 158, 161, 159]. However, since the work in my thesis focuses on developer and system operator usability and security challenges, I do not provide a comprehensive overview of Android security and privacy research. Instead, I point to joint work with Acar et al. [1]. This paper surveys and systematizes the relevant research in this area. Briefly, Android security and privacy research can be split into the following problem areas:

- Permission research attempts to incrementally improve Android's permission mechanisms.

- Alternative approaches to permissions are being proposed and evaluated to improve access control handling for Android apps.

- Webification research addresses security, privacy and usability challenges introduced by Android's move to using web techniques for apps.

- Developer centric research focuses on security, privacy and usability issues caused by developers of Android apps.

- Software distribution research addresses security, privacy and usability issues induced by Android's concept of central software repositories as the de-facto standard to distribute apps.

Since my thesis also investigates usability aspects of app developers and software distributors, the remainder of this section briefly discusses relevant related work.

Egele et al. [59] ask whether app developers use Android's cryptographic APIs in a way that provides typical notions of security such as IND-CPA security. They apply static code analyses techniques to automatically check Android apps on Google Play. Overall, they find that 88% of the apps that use cryptographic APIs make at least one mistake in protecting sensitive user information.

Poeplau et al. [158] conduct a security analysis of Android's external code loading feature that allows developers to load source code at runtime. They apply static code analysis techniques to automatically detect external source loading at runtime and conduct a study of 1,632 free, popular apps from Google Play. 9.25% of those apps' developers implement external code loading in an insecure way, leaving their apps vulnerable to unwanted code injection attacks.

Sounthiraraj et al. [185] investigated the vulnerability of Android apps against Man-In-The-Middle attacks. Their work is based on methodology and results of my work presented in Chapter 6, Page 93. They present and evaluate a tool called SMV-Hunter that allows for automatic large-scale identification of Man-In-The-Middle vulnerabilities in Android apps by combining static and dynamic analysis techniques. SMV-Hunter is evaluated against a set of 23,418 apps downloaded from Google Play finding that 726 apps in their sample were vulnerable. Their results are very similar to the results I report in Chapter 6, Page 93.

2.4 Summary

The first section of this chapter gave a brief introduction into the field of usable security and privacy research. The second section discussed methods applied in the area of usable security and privacy research. In Section 2.3, I summarized relevant background and related work around message encryption, passwords, TLS and appified platforms. The work I present in the following chapters extends the state of the art as follows:

- I present the design and evaluation of a security workflow for message encryption in Chapter 3, Page 33.
- I present a qualitative study with webmasters on the root causes of misconfigured TLS certificates for webservers in Chapter 4, Page 59.
- I analyze the security of password managers for Android and present results of a qualitative study with developers of those password managers to shed light on usability challenges for developers in Chapter 5, Page 75.
- I quantitatively analyze the occurrence of insecure code in custom implementations of TLS certificate validation in Android applications and present results that illustrate usability challenges for Android developers in the context of the secure implementation of TLS in Chapter 6, 93.
- I present results of a qualitative study with Android developers that allowed me to re-design the handling of TLS certificates in the Android ecosystem in Chapter 7, 112.
- I present and discuss a novel and easy to use mechanism that allows providers of central software repositories to prove the authenticity and correctness of the distributed software to their users in Chapter 8, Page 139.
- I motivate more foundational research concerning ecological validity of usable security and privacy research methods in Chapter 9, Page 163.

3 End Users: Encrypting Facebook Messages

Disclaimer: *The contents of this chapter were previously published as part of the paper "Helping Johnny 2.0 to Encrypt His Facebook Conversations"* 8th *presented at the Symposium On Usable Privacy and Security (SOUPS) in 2012 [79] together with my co-authors Marian Harbach, Thomas Muders, Uwe Sanders and Matthew Smith. As this work was conducted with my co-authors as a team, this chapter will use the academic "we" to mirror this fact. The idea and initial concept for this work came from myself. I designed and implemented the message encryption backend: implementing the frontend was joint work with Marian Harbach. The design of the laboratory study was joint work with Matthew Smith and Uwe Sanders: I lead the study execution. Thomas Muders and Marian Harbach assisted in the study execution. Analyzing the laboratory study results was joint work with Marian Harbach. I designed, executed and analysed the interviews by myself. Together with Marian Harbach, Thomas Muders and Matthew Smith, we jointly discussed the work's implications and compiled the paper for publication.*
Software described in this chapter is available at `https://zenodo.org/record/50524`.

3.1 Motivation

The usability of email security for end users has been the subject of research for almost two decades. Whitten and Tygar [201] conducted the first Johnny study in 1999, analysing the usability of PGP5, followed by the more recent evaluations of S/MIME in Outlook Express in Garfinkel and Miller's Johnny 2 study [92] and the re-evaluation of the original Johnny study using PGP9 by Sheng et. al. [179]. In this work, we address the issue of message security in the context of Online Social Networks (OSN) in general and Facebook in particular. Even though the Web 2.0 paradigm is now more than a decade old and OSN sites such as Facebook play a major role in many people's online lives, there has been very little work on the usability of message security in this domain.

Facebook is the largest online social network. In 2010, when Facebook had only 500 million users, Facebook published internal statistics showing that more than 4 billion private messages (including chat messages) were sent every day [1]. Also in 2010, a Gartner study predicted that social networking services would replace emails as the primary vehicle for interpersonal communications for 20 percent of business users [2]. To put these numbers into perspective, Google announced that Gmail had 350 million users in January 2012 [3].

While there are some solutions available to cryptographically protect Facebook conversations, to the best of our knowledge, there is no widespread use of them. Thus, the aim of this work was to find out why this might be the case and what could be done to help end users to encrypt their Facebook conversations. While

[1] `http://techcrunch.com/2010/11/15/facebook-350m-people-using-messaging-more-than-4b-messages-sent-daily/` – last access 13.04.2016
[2] `http://www.gartner.com/it/page.jsp?id=1467313` – last access 13.04.2016
[3] `http://www.email-marketing-reports.com/metrics/email-statistics.htm` – last access 13.04.2016

mechanisms to protect email messaging could in principle be adapted to Facebook conversations in a straightforward manner, previous usability studies show significant problems with the existing email encryption mechanisms. One of this work's goals was therefore to see if the changes brought about by the OSN paradigm might open up new possibilities for a usable security mechanism protecting private OSN messages.

To answer these questions, we conducted multiple studies to evaluate needs surrounding the protection of users' conversations on Facebook and then compared different existing solutions for conversation encryption. Based on these intermediate results, we developed an approach to encrypt Facebook conversations and tested those in two user studies to ascertain whether the solution provided good usability characteristics while at the same time protecting user privacy. The results of the final study show that the OSN paradigm does indeed offer new ways of simplifying message encryption and finding security/usability trade-offs which are acceptable to users.

Interesting related work in the field of email encryption can be found in Section 2.3.2.

3.2 Background

Before exploring how users could protect their Facebook conversations, we conducted a screening study to gain an overview of the level of interest in protecting these conversations. We invited 16,915 students at the Leibniz University Hannover via email to participate in the study. It was introduced as a poll on Facebook privacy. We did not attempt to hide the fact that we were interested in Facebook message privacy, since we explicitly wanted to study those users who would like to protect their conversations. There was no direct reward for completing the poll, however the possibility of a paid follow-up study was stated.

In the poll, we queried some Facebook usage statistics and asked whether or not the participants thought that Facebook could read their private messages as well as whether or not this would be a cause for concern for them. We received 514 responses. Of these, 413 (80.35%) were aware that Facebook was able to access their private messages. When asked whether this concerned them, 263 (63.68%) answered "yes", 78 (18.88%) answered "no" and 72 (17.43%) stated they didn't care. The other 101 (19.65%) participants stated they were not aware that Facebook could read their private messages. When asked whether it would concern them if Facebook could read their private messages, 79 (78.21%) answered "yes", 12 (11.88%) answered "no" and 10 (9.90%) stated they did not care. In total, 342 (66.53%) of the 514 participants stated that they were or would be concerned by Facebook being able to read their private messages.

Since there were users who were concerned that Facebook could read their conversations, we used Google, Bing and Yahoo (in September 2011) and searched for products which could be used to encrypt private messages on Facebook. Encipher.it[4]

[4] `http://encipher.it` – last access 12.04.2016

and uProtect.it[5] were the top hits which could also be installed. The discontinued product FireGPG was not compatible with current browsers[6], so we did not consider it a viable solution that normal users could currently install.

Encipher.it

Encipher.it provides a *bookmarklet* for Firefox, Chrome and Internet Explorer (IE) that is capable of encrypting text in any HTML text area. Thus, to encrypt a Facebook message, the user writes the message text into the Facebook message composer as usual and then has to click on the Encipher.it bookmarklet in the upper browser bar. Next, a popup in the centre of the screen appears and asks the user to "Enter encryption key". Internally, Encipher.it uses AES [154] in Counter Mode [167] for encryption, i. e. the same symmetric key is used for encryption and decryption. To derive a secure symmetric key from the user's input, PBKDF2 [119] is used. After a key is entered, the user must press the "Encrypt" button. The bookmarklet then replaces the clear text in the Facebook message box with an enciphered version that can be sent as normal with Facebook's "Send" button. Key management is left entirely to the user, which means the user must find a secure way of sharing the encryption key with the receiving party.

uProtect.it

Unlike the generic Encipher.it solution, uProtect.it was a third-party service specifically designed for Facebook. The user has to create a uProtect.it account and needs to install the uProtect.it browser plugin. Plugins are provided for Firefox and Google Chrome as well as a bookmarklet for other browsers. After the user has created a new uProtect.it account and installed the plugin, a green bar appears at the top of the browser window and asks the user to log into uProtect.it when the user is on Facebook. Subsequently, orange encryption buttons are placed next to text areas. Messages are encrypted and decrypted by pushing the orange button.

Unlike Encipher.it, key management is handled automatically by the service. Unfortunately, uProtect.it does not provide any information concerning their internal security mechanisms. They do however state that they store the user content on their servers alongside the encryption keys. Thus, they are able to eavesdrop on the users' data, as stated in their Terms of Services[7].

Academic Solutions

Apart from the approaches above, which the average user can easily find on search engines, there are also several academic solutions. Even though these publications focus mainly on the cryptographic aspects of their solutions, each is briefly outlined

[5] `http://uprotect.it` – last access 07.03.2012
[6] `http://blog.getfiregpg.org/2010/06/07/firegpg-discontinued/` – last access 12.4.16
[7] `https://uProtect.it/terms` – last access 07.03.2012

in the following. In 2008, Lucas et al. [140] proposed flyByNight, a prototype Facebook app that encrypts and decrypts messages using public key cryptography. The flyByNight server handles the key management and uses its own database to store the encrypted messages. This is a standalone app which does not protect messages sent via the standard Facebook messaging centre, but rather requires the user to send all messages via the app. Lucas et al. noted that usability would be an issue for future work.

Scramble! [14] is a PKI-based Firefox plugin that can store encrypted social network content either on a third-party TinyLink server or directly at the SN provider. However, as with most PKI solutions, key management is an issue, since it relies on PGP mechanisms and must be dealt with by the user. When sending encrypted content, the user composes a message with the Facebook UI and selects the text he wants to encrypt, whereupon Scramble! requires the user to choose the contacts to encrypt the content for manually. The encrypted text or a TinyLink URL is then placed into the message composer and can be sent through the regular UI.

Another approach was taken by Guha et al. [104], who use shared dictionaries to map different "atoms" of information to a similar, valid piece of information. For example, Alice's address would be randomly replaced by Bob's, according to some mapping key. Their NOYB prototype can hide the fact that content is being protected but also necessitates key exchange using email. Additionally, reusing other users' information can have privacy implications of itself.

Baden et al. [11] present Persona, a privacy-enhanced social network platform, using public key cryptography and attribute-based encryption (ABE). They acknowledge the need to integrate their new service with the popular networks and demonstrate a prototype that provides their services as a Facebook application. They argue that existing SN apps can be gradually migrated to use the Persona platform, at least for storage. Using the Facebook API, it is however not possible to access the messaging service. The Persona user Interface and workflow for sending confidential messages is not explicitly described.

Anderson et al. [6] and Dodson et al. [54] present concepts to use rich-clients as a way to improve privacy. The SN provider is reduced to a mere content distribution server while the client handles cryptography and information semantics. This approach would require a user to migrate to another SN and change the interaction patterns, which is a different scenario from that this paper addresses.

3.3 Exploratory Phase

Unlike in the related email-based studies, where relatively mature and stable implementations of PGP and S/MIME were available and could be studied directly (cf. Section 2.3.2, the solutions for Facebook are partly general purpose encryption products which can also be used with Facebook or early academic prototypes and niche products with usability issues which stem more from implementation limitations than design issues. For this reason, we decided to extract the design decisions and build mockups to study the basic building blocks and their usability issues. A

further reason for choosing this abstract approach over a direct product evaluation was that the two available solutions, Encipher.it and uProtect.it, differ in several key areas, which would have made it very difficult to judge which features made the one more usable than the other. Thus, we extracted core features of the above solutions to study the usability of encryption for Facebook conversations.

Three features are particularly well suited to distinguish the above approaches: encryption UI, key management and integration. For the encryption UI, some solutions require the user to trigger the encryption process manually by activating a bookmarklet or pressing a button, others trigger encryption automatically. The different key management options require the user to get involved in the key management process by manually sharing or selecting keys, while other solutions automate this issue. A further feature is integration. Some solutions require the user to send private messages via a completely separate UI instead of Facebook's standard UI, while others integrate their solution into Facebook. In order to keep the study design as simple as possible, we chose to focus on integrated solutions, because we believe it is better not to force the user to leave the normal Facebook UI. Table 3.3 gives an overview of the values for the two remaining variables in the two real-world solutions. Based on this extraction, we built four mockups, described in the following section, which were then used for the laboratory study.

	Encipher.it	**uProtect.it**
Key Management	manual	automatic
Encryption	manual	manual

Table 3.1: A comparison of key management and encryption/decryption concepts applied by Encipher.it and uProtect.it.

To evaluate the different interface and workflow concepts for sending encrypted Facebook messages as discussed above, we built mockups using Greasemonkey[8]. The mockups allowed us to test the independent variables shown in Table 3.3 in the context of sending encrypted private Facebook messages. Screenshots of the mockups are shown in Figures 3.1 and 3.2.

Mockups

Table 3.3 gives an overview of which condition is dealt with in which task. Based on these tasks, we created mockups 2 – 5 for conducting our user study.

Figure 3.1 shows mockups 1 and 3 corresponding to manual encryption combined with both manual and automatic key management. In the case of manual encryption with manual key management, the user enters the message text as usual (Step 1). The user must then click the new "Encrypt" button. A popup asks the user for an encryption password with which the message is encrypted (Step 2) and the resulting ciphertext is placed in the message box. The user can then send the message using the original "Send" button (Step 3). The encryption password must be shared with

[8] `http://www.greasespot.net/` – last access 13.04.2016

Task	Interface	Encryption	Key Management
TBase	Facebook	None	None
TEmKm	Mockup 1	Manual	Manual
TEaKm	Mockup 2	Automatic	Manual
TEmKa	Mockup 3	Manual	Automatic
TEaKa	Mockup 4	Automatic	Automatic

Table 3.2: Properties of the tasks in the lab study.

the recipient manually. This corresponds to the Encipher.it workflow. All steps are repeated for every message sent.

In the case of manual encryption with automatic key management, the key management model from uProtect.it is used to replace the manual key management of Encipher.it. This means that Step 2 only needs to be executed once per Facebook session, since the password can be cached locally and the entered password does not need to be shared manually with the recipients.

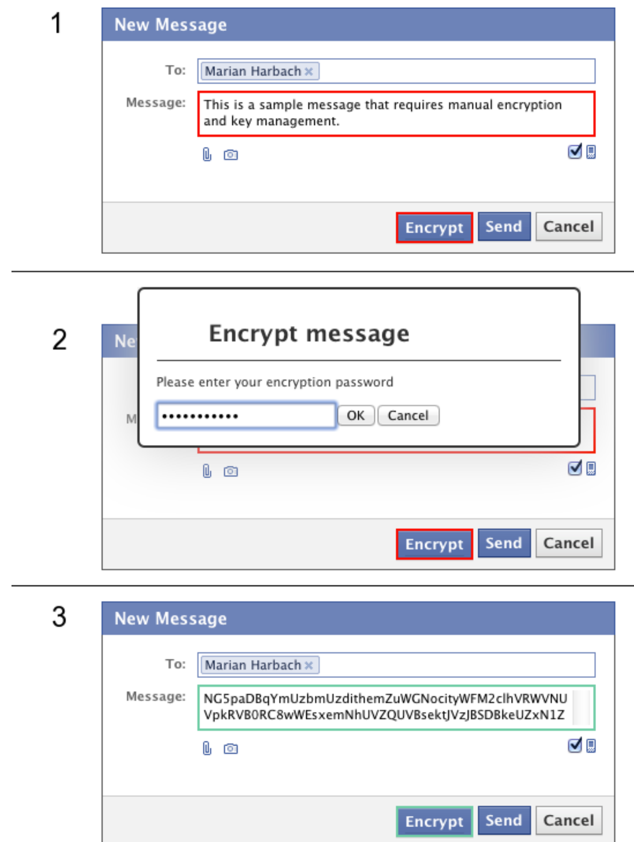


Figure 3.1: The three steps in mockups 1 & 3

Figure 3.2 shows mockups 2 and 4 corresponding to automatic encryption combined with both manual and automatic key management. With these mockups, the user does not need to manually trigger encryption. Rather, when the Facebook

“Send” button is pressed (Step 1), encryption is triggered automatically. In the case of manual key management, the user needs to choose an encryption password for each message to be sent and share it with the message recipient manually (Step 2). In the case of automatic key management, the user only needs to enter the password once per Facebook session, as in the uProtect.it workflow. In order to offer a similar amount of visual feedback as in mockups 1 and 3, the message is not sent instantly after completion of Step 2. Instead the ciphertext is shown in the message composer’s text area with a spinner animation for two seconds to visually indicate successful encryption after which the message is sent (Step 3).

We also added red and green visual security indicators to the text area and the “Send” button of mockups 2-5 as a visual aid, as suggested by Egelman et. al. [62] and Maurer et. al. [148].

In addition to mockups 1 to 4 described above, we built mockup 0 without any modifications of the Facebook message composer to serve as control condition.

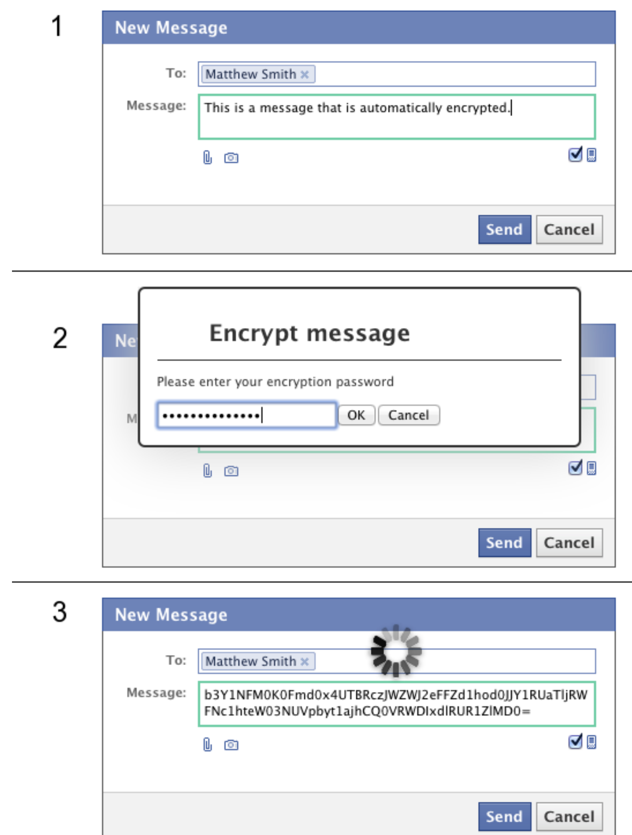


Figure 3.2: The three steps in mockups 2 & 4

Laboratory Study

Based on the concepts and mockups presented above, we conducted a laboratory study. The goal of the study was to evaluate the basic building blocks of a message

encryption mechanism. We therefore tested the influence of manual vs. automatic encryption and manual vs. automatic key-management on usability, acceptance and perceived security. We also wished to find out what role password or key recovery plays in the acceptance of an encryption mechanism. Depending on the cryptographic principle used, the loss of the encryption key can result in complete loss of the encrypted data in case the key cannot be recovered and as a result decreases both the acceptance and the utility of a solution.

During the study, each condition (cf. Table 3.3) was dealt with in a separate task with a separate mockup.

Technical Setup

The study took place in our usability lab, where we had set up a PC with Firefox 9, Greasemonkey, the mockups and a webmail interface for each participant. We created artificial Facebook accounts and email addresses, so that the participants did not have to use their real accounts and data. The mockups simulated sending private messages, rather than actually sending the messages which might have accidentally triggered the anti-spam protection of Facebook, resulting in blocked accounts. However, we did ensure that we simulated the behaviour of Facebook's standard message composer, so that participants would not notice that messages were not actually sent.

Participants

For this study, we were interested in participants who would potentially want to use an encryption mechanism to protect their Facebook conversations. Educating or motivating participants who are not worried about their conversation's privacy was outside the scope of this work. We randomly selected test candidates from the poll participants (cf. Section 3.2), who met the following criteria: they needed to be concerned that Facebook could access their private messages and needed to use Facebook at least on a weekly basis. We excluded infrequent Facebook users to minimise the risk of technical difficulties when using Facebook. Finally, we excluded computer science students to avoid bias based on technical skills and possible familiarity with encryption mechanisms.

This left us with 291 possible candidates, from whom 100 were randomly selected for the study. 96 of these attended their appointed slot. Each participant received a compensation of 10 Euros. All participants were students from the Leibniz University Hannover. Table 3.3 gives demographics for the participants.

Ethical Considerations

The study was conducted in Germany and thus was not required to pass an IRB review. Nevertheless, our studies complied with the strict German privacy regulations. We did not use the participants' real Facebook accounts and all data was collected anonymously. After the study, the participants were debriefed and any questions regarding the study were answered.

	N=96
Gender	
Male	44
Female	52
Age	M=22,SD=2
< 20	12
20 - 25	69
> 25	15
Facebook Membership	
6 months	7
1 year	16
2 years	37
longer	34
don't know	2
Facebook Password Loss in The Last 12 Months	
not once	79
once	9
twice	3
three times	2
more than three times	3
Facebook use	
several times per week	10
< 1 hour per day	27
1 - 2 hours per day	41
2 - 4 hours per day	15
more than hours per day	2
Facebook Friends	M=207,SD=130
50 - 100	20
101 - 150	24
151 - 250	28
251 - 350	13
> 350	11
Facebook Messages / Week	M=24.35,SD=46.68
< 10	45
10 - 20	27
21 - 30	8
> 30	16
Facebook Chat Use	
several times a day	15
daily	23
weekly	28
less frequent	17
not at all	13
Prior Contact With Encryption Mechanisms	
yes	33
no or don't know	63

Table 3.3: Demographics of the laboratory study participants.

Procedure

The participants were informed that they would be testing five different technologies to encrypt Facebook conversations. To avoid bias, we explained that the technologies were not built by us and that we were testing the technologies, not the participants.

Each participant was watched by a study monitor, who measured the time needed to complete each task and noted errors. The monitor was allowed to assist with the browser tabs and the webmail program, but no help or information was given concerning the mockups or the tasks themselves. The next section outlines the basic structure of each task (cf. Table 3.3).

Tasks To keep the design simple, all tasks were focused on encrypting and sending private Facebook messages to three different friends (Jan, Vanessa and Heike). The decryption process is analogous to encryption and was therefore not tested explicitly.

Handouts were given to the participants which explained the procedure of sending an encrypted message with the given technology. The messages to be sent were as follows:

To Jan: Hello Jan. Please transfer the money to my bank account, account number 123456 and sort code 100200.

To Vanessa: Jan has transferred the money to my bank account.

To Heike: Hi Heike. Have you transferred the money yet?

Since all participants had a self-reported interest in protecting their Facebook conversations from unauthorised access, we chose sample messages which contained financial information with the aim of inducing a similar wish for privacy in all participants.

Task TBase was the control group task. Participants were asked to send the messages using the normal Facebook message composer. The task was used to get a baseline for error rates and speed of the individual participants. Like in the other tasks, the participants were told that their messages were encrypted. In contrast to TEmKm to TEaKa, message encryption was not featured explicitly, but included in the regular sending process without visual indicator or actions. The control task therefore additionally lends itself to examine whether or not the participants would accept and trust a mechanism that provides "invisible" security.

During the manual key management tasks (TEmKm and TEaKm), the participants needed to use the webmailer to send an arbitrarily chosen key to the corresponding recipients out-of-band. Using webmail is of course not the optimal out-of-band solution in terms of security. However, since the study's focus was on the Facebook UI and not the out-of-band communication capabilities of the participants, it was used as a mechanism which would cause little technical trouble during the study. In a real world setting additional problems could arise here.

In the automatic key management tasks (TEmKa and TEaKa), only the first message required the participants to enter their password. The password was cached for the rest of the session.

The only difference between the manual and automatic encryption tasks is that the "Encrypt" button needs to be pushed before sending the message.

Study Design

Since the study encompassed reading and comprehension, we chose a within-subjects design [137]. To minimise the bias of the learning effect, we also chose a random latin square setup, so that each task was equally distributed over each position in the within-subjects design.

In the post-task questionnaires for each of the five tasks (cf. Section 3.3 and Appendix A.1, Page 193), we collected the system usability score (SUS [29], see Appendix A.1, Page 193) as well as additional items concerning the end users' willingness to use the corresponding mechanism in the future for private and general messaging ("acceptance"). A final item gauged how well the users felt their messages were protected.

After completing the tasks, the participants were given a final questionnaire (cf. Appendix A.1, Page 193). Apart from gathering demographic information, the questionnaire also presented a hypothetical question, asking whether or not the participants would use an encryption method which would render all previous encrypted messages unreadable if they forgot their password. We also asked supporting questions to ascertain the reasoning behind this decision.

Results

Across all cases, we found the highest mean SUS values in TEmKa (86.51) and TEaKa (89.79), as well as in the control TBase (88.20, cf. Table 3.3). TEmKa and TEaKa also received the highest acceptance ratings for both private and general messaging. However, the users felt best protected in TEmKm and TEaKm.

Task	SUS	sd_{SUS}
TBase	88.20	15.32
TEmKm	64.27	18.56
TEaKm	65.86	18.43
TEmKa	86.51	11.43
TEaKa	89.79	14.20

Task	a_{priv}	sd_{priv}	a_{all}	sd_{all}	sf	sd_{sf}
TBase	2.62	1.438	2.67	1.449	1.57	0.778
TEmKm	3.19	1.439	1.87	1.136	3.49	1.133
TEaKm	3.35	1.421	1.87	1.117	3.42	1.158
TEmKa	3.87	1.259	2.91	1.437	3.20	1.148
TEaKa	3.92	1.319	3.30	1.415	3.23	1.174

Table 3.4: Mean usability (SUS) and acceptance for private (a_{priv}) and all messages (a_{all}), as well as security feeling (sf) across tasks.

We aggregated the SUS [9]

[9] The System Usability Scale (SUS) provides a reliable tool for measuring the usability. It consists of 10 questions and has a score between 0 (poor usability) and 100 (great usability). A SUS score above a 68 would be considered above average and anything below 68 is below average. [28]

and acceptance ratings for tasks with (non-)automatic key management (TEmKm-TEaKm and TEmKa-TEaKa) and encryption (TEmKm-TEmKa and TEaKm-TEaKa) respectively. Normality tests indicated significant or almost significant deviations for these scores and ratings, since the distributions were cut off at the upper score-interval boundary. Therefore, we used the non-parametric Wilcoxon Signed Ranks test to analyse the scores and ratings. We found a significant difference in SUS for automatic key management ($Z = -8.102$, $p < .01$) and automatic encryption ($Z = -3.230$, $p < .01$). We found similarly significant differences in acceptance ratings for all messages with respect to automatic key-management ($Z = -6.884$, $p < .01$) and automatic encryption ($Z = -2.692$, $p < .01$). Acceptance for sending private messages differed significantly for automatic key-management ($Z = -3.644$, $p < .01$) but not for automatic encryption ($Z = -1.637$, $p = .102$). We therefore conclude that an optimal workflow would use automatic key management while automatic encryption did not have a significant impact in the study.

To test how fear of losing data influences the need for password recovery, we divided the participants into those who stated that they were worried or very worried about losing all their old messages or forgetting their password (group A, $n = 49$) and those who were not (group B, $n = 47$), using top-2-box scores of a 5-point Likert scale. We found a significant difference between group A and group B using a Chi-Square Test concerning whether or not they would use a mechanism without password recovery ($\chi^2_1 = 18.383$, $p < .001$) and whether or not they would prefer a mechanism with password recovery ($\chi^2_1 = 10.341$, $p < .001$). In group A, 72.3% would not use a mechanism without recovery and 78.7% would prefer a mechanism with password recovery, while in Group B these figures were 28.6% and 46.9% respectively. Hence, we believe that password recovery is desirable for users, especially for those who worry about forgetting their password.

To test the correlation between the perceived usability and the stated acceptance of a message protection mechanism, we used Spearman's rho and found significant values in all five tasks (see Table 3.3). However, the correlations are only weak to medium and therefore merely suggest that higher usability correlates with higher acceptance. We investigated this issue further in the interviews (cf. Section 3.5).

Task	$\rho_{private}$	p	ρ_{all}	p
TBase	.253	$< .05$	.260	$< .05$
TEmKm	.554	$< .01$	.361	$< .01$
TEaKm	.466	$< .01$	.249	$< .05$
TEmKa	.533	$< .01$	.407	$< .01$
TEaKa	.530	$< .01$	.507	$< .01$

Table 3.5: Spearman's correlation between usability and acceptance for private/all messages across tasks.

In order to investigate the perceived level of protection across mechanisms, we ran a Friedman test on the participants' answers concerning their perceived protection in tasks TEmKm through TEaKa. We found a highly significant difference in the mean ranks ($\chi^2_3 = 15.947$, $p < .001$). The top-2-box scores show that in tasks

TEmKm and TEaKm 54.2% of the participants felt well protected and in TEmKa and TEaKa only 41.7% and 40.6% felt the same way. We therefore suspect that the complexity of a mechanism – in this case creating individual encryption keys for each recipient and distributing them manually – heightens a user's subjective sense of security. However, we could not find any meaningful correlations to support this.

It is noteworthy that only 2% of the participants felt well protected in the control task. Even though they had been told that the mechanism presented in TBase would protect their message, they apparently placed little faith in this statement. While this could be due to their familiarity with Facebook, we also suspect that an entirely invisible and effortless protection mechanism does not generate a feeling of security and is not trusted by users. This is an interesting question, since "invisible" security is often claimed to be a desirable feature. However, our results suggest that trust in the mechanism could be a problematic issue. This study was not set up to analyse this observation further, but this issue might be worth a dedicated investigation in the future.

To analyse who was perceived to be the biggest privacy threat, we also asked participants to rate how easy it would be for different entities to access their Facebook conversations on a 5-point Likert scale. Facebook employees and hackers were perceived as having the easiest access to that information: 87.5% and 84.4% said that they thought it would be easy or very easy for these actors to access their private messages, followed by the government of the USA (62.5%), advertising agencies (49.0%) and the German government (35.4%). Only 12.5% believed that it was easy or very easy for their friends to access these messages. Additionally, we wanted to know how motivated the participants believed these entities would be to access their messages. Advertising agencies were believed to be the most eager (70.8%). Facebook (44.8%), Hackers (29.2%), the US government (28.1%) and the German government (25.0%) are believed to have less motivation to access private messages. Friends were believed to be the least motivated (18.2%). Finally, we asked how bad the participants would feel if these entities accessed their private messages. 55.2% would find it bad or very bad if friends could access private messages not intended for them. For all the remaining actors, the participants almost unanimously agreed that access to their private messages would be bad or very bad (82.3% to 90.6%).

3.4 Usable Facebook Message Encryption

Our aim was to create a security system with good usability which addressed the concern that Facebook and potentially other third parties could read private messages sent via Facebook. Considering the findings of the lab study, we based our design on the interface and workflow of mockup 5. While the manual key management mockups 2 and 3 created a higher security feeling, they also had significantly lower acceptance and usability scores. We chose mockup 5 over mockup 4 due to the higher acceptance and usability scores of 5. While mockup 4 included some manual operations, there was no significant difference in the perceived level of protection between mockup 4 and 5.

One of the key decisions for our implementation concerns the use of a PKI. Based on the fact that previous Johnny studies have shown that PKI based key management and message protection has severe usability problems, we decided to avoid the use of a PKI and opt for a simpler approach. Hence, our implementation of message encryption for Facebook addresses confidentiality and integrity, using the non-cryptographic message authentication offered by Facebook. By dispensing with digital signatures, it was possible to create a simpler overall system. This is a security/usability tradeoff. Since the main aim of protecting users' private messages from entities which are currently able to read them can be achieved with confidentially alone, the reduction in complexity was the deciding factor in this matter.

However, we would like to briefly discuss message authentication in the social web. The use of Facebook brings about some interesting changes in certain aspects of the message authentication landscape. While emails can be easily forged and are also used to initiate communication with unknown communication partners, in the social network context much of the communication over Facebook is conducted in the context of "friendship-connections" which are established a priori and filled with additional information. This reduces the need for authentication on the message level to a certain extent. While there are social-engineering-based attacks, in which users can be duped into falsely believing a message originated from a friend, we believe these are currently less relevant than for example email-spoofing attacks. This makes the lack of message level authentication less problematic for a social web context than for emails. However, this last statement is speculative and needs to be the focus of a separate study.

The choice to offer only confidentially also enabled us to offer a key recovery feature that allows users to recover their encryption passwords. For this, we opted for a service-based approach offering confidentiality as a service which we named FBMCrypt. Special attention was paid to creating a service that does not enable the FBMCrypt provider to access the private messages, but allows automated key management at the same time. To enrol in the service, a user needs to register and bind his Facebook account to FBMCrypt. This will be illustrated in the following.

Registration

For registration with the FBMCrypt provider, we chose a simple username/password authentication scheme, since this method is a well-known scheme to Web 2.0 users. Although passwords are not the strongest authentication credentials, they enjoy widespread application and are the most widely accepted concept by online users [111]. The registration process relies on Email-Based Authentication and Identification (EBIA) [93], the most prevalent authentication scheme for online accounts.

Account Binding & Browser Plugin

Once the registration process is complete, the user needs to bind his Facebook account to the newly created FBMCrypt account. This is initiated by clicking a button to log into Facebook using Facebook's Social Plugin API. After Facebook

has confirmed the authentication – through Facebook's OAuth mechanism – the user agrees to allow the FBMCrypt provider to see the email address registered with Facebook. The FBMCrypt provider uses this email address to send a second validation link, which establishes that the currently logged-in FBMCrypt user also has access to the Facebook account in question and can furthermore read email sent to that account. This process only proves that the current FBMCrypt user has access to the Facebook account, but does not give the FBMCrypt provider access to the Facebook account. After the successful binding of a Facebook identity to a FBMCrypt account, the user is subsequently able to use the FBMCrypt provider's services with the bound Facebook identity.

The user finally needs to install a browser plugin which handles the actual encryption and decryption of the messages. Similar to the mockups, our prototype plugin uses a Greasemonkey user-script, which is easy to install. The plugin communicates transparently with the FBMCrypt provider and handles all the cryptographic operations for encryption and decryption without requiring any further user involvement, apart from entering the FBMCrypt password once per session.

Sending an Encrypted Message

The FBMCrypt plugin automatically checks if the recipients of a message are registered with the FBMCrypt service. If they are not, the message cannot be encrypted and can only be sent in the clear. We modified the message composer to make the user aware of an unencrypted message exchange, as illustrated in Figure 3.3.

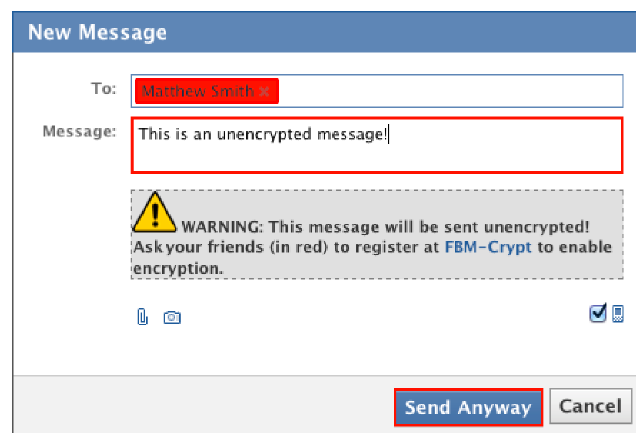


Figure 3.3: The modified message composer in case a message will be sent unencrypted.

If the recipients are enrolled with the FBMCrypt service and the "Send" button is clicked, the FBMCrypt plugin connects to the FBMCrypt service over a secure HTTPS connection. The plugin authenticates the user to FBMCrypt by sending the previously bound Facebook ID and the user's hashed FBMCrypt password.

The plugin then triggers FBMCrypt' automatic key management (cf. our previous work in [78]) procedure and encrypts the message.

Before actually sending the encrypted message via Facebook, the plugin prefixes *##caasfb##* to the message to allow for automatic decryption on the recipient's side.

Reading an Encrypted Message

To read an encrypted message, decryption is integrated into the usual workflow as transparently as possible. The plugin automatically analyses the messages site and searches for FBMCrypt-encrypted ciphertexts. If one is detected, the plugin checks the user's credentials, authenticates him to the FBMCrypt service and triggers automatic decryption. Figure 3.4 shows the message view with a decrypted and encrypted message.

Figure 3.4: Cleartext and ciphertext of a FBMCrypt protected message

We provide a more detailed technical description of the confidentiality as a service backend and the encryption mechanism as well as a security discussion in [77, 78]. The following focuses mainly on the usability aspects of the message encryption mechanism.

3.5 Evaluation

Case Study

The registration and account binding procedure (cf. Sections 3.4 and 3.4) was designed to enable FBMCrypt to fulfil the requirements derived from the laboratory study. Since these steps form the basis for the rest of the system and poor design could potentially deter users from the outset and make further development unnecessary, we conducted an initial study of the registration and binding design to ensure the usability of our concept, before proceeding with the development of the rest of the system.

We ran a field study with 20 participants (all undergraduate students, 9 females, 11 males, with an average age of 23) and asked them to register an FBMCrypt account, bind that to a Facebook account and install the plugin. Since this was only a simple ten minute task to eliminate early issues in the design phase, we recruited students randomly on campus. We asked them if they were Facebook users and

interested in participating in a ten minute scientific study that was about a security mechanism for their private Facebook messages in exchange for some candy bars.

The technical setup was similar to the lab study: we provided a laptop with Firefox 9 and Greasemonkey and asked them to log into their Facebook and email account. We deleted all browsing data after the experiment ended. For this initial study, we were interested in the time needed to set up a working FBMCrypt plugin installation and corresponding error rates.

All participants were able to successfully create an FBMCrypt account, bind the account to Facebook and install the FBMCrypt plugin for Facebook. On average, the entire process took 3 minutes and 8 seconds, with a range of 90 seconds to 6 minutes and 18 seconds. Since no problems with the installation and binding process were identified, the design was integrated into the rest of our approach.

Interviews

To evaluate the usability of the proposed FBMCrypt service (cf. Section 3.4) as a whole, we conducted a final qualitative study with 15 participants in which the entire process was evaluated in conjunction with an online survey and a semi-structured interview. During the study, one interviewer and one assistant were present.

Participants

We randomly recruited the participants from the same pool of users that we used for the laboratory study, excluding those that had already taken part. There were 6 male and 9 female participants. On average, their age was 22 ($sd = 3.39$) and 13 of them had been using Facebook for more than a year. Three of them had forgotten their Facebook password at least once and 14 used Facebook for at least one hour per day. They had 233 Facebook friends on average ($sd = 125$) and all of them sent at least five private Facebook messages per week. More detailed demographics can be found in Table 3.5.

The technical setup and procedure was analogous to the laboratory study, except that during the task participants were audio recorded and asked to “think aloud”. To test our encryption mechanism for Facebook conversations, all participants were asked to fill out an online survey, complete a task involving three subtasks and participate in a semi-structured interview. The entire study lasted between 28 and 44 minutes ($mean = 33$, $sd = 4$).

Task

Firstly, the participants were asked to register for the FBMCrypt service.

After the EBIA procedure (cf. Section 3.4) was completed and a new FBMCrypt account created, this account had to be bound to the Facebook account provided for the participants (cf. Section 3.4). Successfully binding the accounts allowed the participants to install the encryption plugin as the last step of the first subtask. After the plugin was installed and operational, they started with the second subtask.

	N=15
Gender	
Male	6
Female	9
Age	
< 20	3
20 - 25	9
> 25	3
Facebook Membership	
1 month	1
6 months	1
1 year	3
2 years	5
longer	5
Facebook Password Loss in The Last 12 Months	
not once	12
once	2
more than three times	1
Facebook use	
$<$ 1 hour per day	6
1 - 2 hours per day	8
several times per week	1
Facebook Friends	
50 - 100	3
101 - 150	2
151 - 250	4
251 - 350	3
> 350	3
Facebook Messages / Week	
< 10	7
10 - 20	4
21 - 30	3
> 30	1
Use Harddisk Encryption	2
Heard of AES	1

Table 3.6: Demographics of the interview participants.

Here, the participants were asked to have an encrypted Facebook conversation with the assistant. The conversation was initiated by the assistant, who sent the following message: *Hi <participant's first name>, what is your major at university?* Next, the participant was asked to answer the question as he would usually do when sending a Facebook message. The assistant sent a new encrypted message: *Sounds interesting. Do you happen to know what AES is?* Depending on the participant's answer, the assistant either answered: *No problem, thanks anyway and have a nice day!* or *Thank you very much, you really helped me. Have a nice day!*

Finally the last subtask was to send another pre-defined Facebook friend an arbitrary message. This friend, however, was not yet registered with FBMCrypt. In order not to bias the participants, we did not indicate that this message would be sent in an unencrypted fashion.

Interview

The interview component of the final study was conducted as a semi-structured interview. The framework of themes to be explored during the interview encompassed a usability evaluation of the encryption service registration, binding and plugin installation, sending/reading an encrypted message, the perceived security and reasons for or against the proposed password recovery mechanism (cf. Appendix A.2, Page 195).

Data Analysis

We transcribed the audio recordings of the interviews. Trends were identified and answers grouped into categories for each question in the interview.

Results

This section presents the findings of our final study. Firstly, we describe the reception of the FBMCrypt registration, binding and installation process, such as how users feel about creating an extra FBMCrypt service account, choosing a different password than the one for their Facebook account and installing the plugin. Section 3.5 describes usability findings while sending and receiving encrypted Facebook messages. Section 3.5 describes the perceived security while using FBMCrypt. The last subsection discusses the participants' attitudes to the key recovery feature.

We refer to the participants as $P01, P02, \ldots, P15$. $P14$ stated that he already used a mechanism to encrypt his Facebook messages. No participant used any encryption for their email, though $P02$ and $P06$ already had experience with software to encrypt their hard disks. $P05$ and $P07$ did not know whether they used any software to encrypt their data and the rest stated they did not use any encryption mechanism. We asked the participants to rate their computer expertise by telling us how they handle computer problems they or their friends have. $P02$ and $P15$ self-reported their computer expertise as high, $P06, P09$ and $P13$ as medium and the rest as low.

N=15	**mean**	**sd**
I'm sure that I used the mechanism correctly	3.93	1.03
I would send sensitive messages with this mechanism in the future	4.06	0.96
I would send all my messages with this mechanism in the future	3.46	1.06
I have the feeling that my messages are now well protected	3.53	1.06
I found applying the encryption mechanism irksome	1.67	0.89

Table 3.7: Case study post-task survey. (1=Strongly disagree; 5=Strongly agree)

The mean values we found using the post-task survey were slightly better than those in the lab study (cf. Section 3.3), but there were no statistically significant differences. Table 3.5 gives a descriptive overview of the survey answers.

In general, after creating a new FBMCrypt account, installing the browser plugin and actually encrypting messages, participants were confident that they were using the system correctly, would like to send future messages protected by the FBMCrypt

mechanism and did not perceive the encryption as obstructing their workflows. The following subsections will discuss the results of individual aspects of our final study.

The Setup Process We asked our participants about their impressions of the registration, binding and installation process of the FBMCrypt service and plugin. Additionally, we asked the participants to compare the process with creating a Facebook or an email account. During the task, all 15 participants were able to successfully register an FBMCrypt account, bind this account to the provided Facebook account and download and install the plugin. On average, the complete setup phase took 3:51 minutes ($sd = 51s$).

In the interview, we first asked the users to rate the account registration process itself. Overall, the registration process was described as *"easy"* and *"appropriate"* in the context of online service accounts. P02 said *"I would describe the effort involved in setting up such an account as relatively small. I think it took me about 30 seconds – if it really helps to protect my messages this is definitely worthwhile."* Only P07 described the registration process as *"complex – just like setting up my Facebook account. For that I asked my boyfriend to help me to setup the account."* and described the registration effort as *"too high"*. Two participants added a condition to their rating and said the effort would be acceptable if the service really provided protection for their data and was not a subsidiary of Facebook. Eight participants described the FBMCrypt registration process as *"more pleasant"* than creating a new Facebook account, because *"they did not want to know so much information, such as my birthday or phone number"*.

All participants described the fact that the FBMCrypt password needed to be different to the user's Facebook account as *"understandable and unproblematic"*. P10 said *"using two different passwords for Facebook and the encryption service is obvious, because every hacker that knows my Facebook password also would try this password to login to my FBMCrypt account to read my conversations. And if both passwords are the same the encryption would be pointless."* 11 participants stated they used different passwords for online services that they either memorise or write down. The rest used three to six different passwords for all their online accounts. In general, the participants stated that they rarely forgot or lost their passwords – only *"passwords for services I rarely use"* (P06, P08, P10, P12) were liable to be forgotten. P12 added: *"but in this case there is this great 'lost password' button I already had to use a couple of times"*. In contrast to other online services, Facebook passwords were forgotten less frequently. Only P06 had once forgotten her Facebook password. The participants estimated that their FBMCrypt password would be as *"safe"* as their Facebook password, because encryption service is so *"closely linked"* to their Facebook account: *"If I have to enter my FBMCrypt password each time I read my Facebook messages, I am pretty sure not to forget it [because I use Facebook so often]"*.

The account binding process was rated as *"coherent"* and *"appropriate"* in general. Three participants had security concerns during the binding process. Two participants (P02 and P12) falsely identified the binding process as a Facebook App, which they distrusted in general and did not use. P02 said: *"in general I have an aversion*

to Facebook apps, because I don't know what information they secretly use". During the "think aloud" phase, P05 said she would not have downloaded and installed the plugin on her own laptop because *"my boyfriend told me not to download anything from the Internet"*.

Encryption/Decryption We asked the participants to rate the process of sending and receiving encrypted as well as unencrypted private Facebook messages. Two participants (P11 and P14) would use FBMCrypt to send *"sensitive"* messages but not *"smalltalk messages that are not very private"* (P11). The other participants said they would like to send *"all messages with FBMCrypt enabled, if possible"*. The *"if possible"* condition had two different manifestations – two participants (P01 and P08) would use it for all their messages if they felt that the service *"really was secure"* and the second group of participants would send all their messages with FBMCrypt if the service gained widespread adoption and their friends used it as well.

We asked the participants to attribute properties to the process of sending and reading (un-)encrypted messages with the FBMCrypt plugin. The participants gave answers such as *"uncomplicated, simple, secure"* and *"as easy as without the service"*. P15 stated: *"I thought there would be annoying popups and I really liked that none appeared."*. P10 described it as an *"invisible assistant"*. Next, the participants were asked to describe the interface for sending and reading (un-)encrypted messages. Two participants (P04 and P14) did not perceive any difference compared to the normal interface.

The green and red borders, indicating encrypted and unencrypted messages respectively, were thought to have two different meanings. Six participants interpreted the different border colours as *"a green border stands for secure messages"* and *"a red border stands for insecure messages"*. Four participants said the green border indicates their conversation partner *"also has the programme installed"* while a red border indicates the conversation partner is not an FBMCrypt user. Six participants noticed that the ciphertext was displayed before an encrypted message is sent or an encrypted message is decrypted. Five participants stated they saw that the messages were encrypted *"because of the jumbled up text that was displayed"*. Four other participants described the ciphertext as *"jumbled up text"* but did not recognise it as ciphertext. However, the presence of ciphertext did not disturb them in their workflow or caused concern.

All but one participant (P13) would recommend FBMCrypt to their friends to enlarge the group of people they can securely communicate with. We also asked the participants if they would be willing to pay money to encrypt their Facebook conversations. Four participants said no – while P01 would not pay money for such a service for herself she said: *"if I had children who used Facebook, I would pay money to protect their privacy."* All the others were willing to pay *"a small amount of money"*. Five participants preferred a single payment: *"the price should be similar to an iPhone App"*. Seven participants stated that they could imagine paying a *"monthly fee"* ranging from 5 to 10 Euros.

Perceived Security We were interested to see if the application of FBMCrypt affected the perceived security of the participants. Firstly, we asked the participants whether they would send messages which are more confidential via Facebook if FBMCrypt were used. None of the participants affirmed this. All of them said they could not sufficiently *"trust"* the encryption mechanism at this point because they could not verify if it was functioning properly. So while they were all satisfied with the usability and would use FBMCrypt for their current messages, messages with a higher level of confidentiality would still not be sent over Facebook.

Participants' views on this can be divided into two groups: Four participants were sceptical by default and would not trust computer systems in terms of data security without more detailed knowledge. P06 said: *"in the Internet, you can download a program to crack everything, so I do not trust computer systems in general. This is similar to online banking. Although I see this little lock in my browser, I am not really sure that no one can steal my data [because I think anyone could put a lock like that in my browser bar]"*. The second group of 11 participants did not trust the mechanism because they did not know *"if it really works"*. P02 (who also falsely identified the FBMCrypt plugin as a Facebook App) said: *"I really cannot say if the program does what it purports to do. I mean, any app could probably draw a green border around my message to simulate security. I would need some proof of security."*.

To investigate why the participants were so sceptical and to ascertain what could be done to alleviate their doubts, we asked why they did not trust the mechanism. They all said they could not verify whether or not the mechanism really did what it said and needed *"proof"*. When we asked what kind of proof that might be, there were three types of answer. Three participants said they would trust *"reports in specialist magazines"*. Participant P10 said his trust would depend on the operator of the FBMCrypt service: *"I would trust the encryption service if it was operated by a university or a nonprofit organisation that campaigned for privacy on the Internet."*. The remaining participants would trust the judgement of *"friends that know computers well"*.

We also asked the participants if the application of FBMCrypt influenced their perception of privacy. Eight participants stated that they had a more positive perception of their message privacy when using FBMCrypt. Two participants (P04 and P09) referred to the displayed ciphertext before sending a Facebook message as the reason for their changed perception. P15 said that *"installing the extra program made me feel better"*. P05 said: *"entering a second password results in a double protection for my messages which makes me feel more secure."*. The rest of the participants said that applying FBMCrypt did not improve their perception of privacy.

Password Recovery To get a better understanding of the trend towards preferring a mechanism that allows for key recovery (cf. Section 3.3), we asked the participants if they would use the FBMCrypt mechanism if *"losing the password resulted in not being able to access messages that were encrypted with the FBMCrypt mechanism"*. Eleven of the participants would not use the service if losing the password resulted

in losing their messages. P15 said: *"I sometimes use Facebook to share important job-related information. [...] I would definitely need a recovery mechanism because losing access to my data would be disastrous."*. Five participants suggested integrating a password recovery mechanism similar to Facebook's. P15 wanted a *"more secure recovery mechanism with telephone verification in addition to a confirmation email."* Only one participant stated that she would not use a mechanism with password recovery because of security concerns (P12): *"This would be much less secure, because a hacker who has access to my email and Facebook account can then also decrypt my Facebook messages."* Three participants of this group said they *"never read old Facebook messages"* (P05 and P03) or they would ask the conversation partner about the content (P6), hence they were unconcerned about losing access to their previous conversations.

We also asked the participants if they would prefer a password recovery mechanism. Twelve participants would prefer a password recovery mechanism. Most said that they could not guarantee that they would never forget their password (even if *"from an empirical point of view, my risk is very low"* – P05) but they did rely on being able to access their archived messages. Again, P12 would not chose a password recovery mechanism because of security concerns.

Discussion

During the task, we focused on the usability of the FBMCrypt service and the participants' willingness to use the mechanism. The results show that most participants rated the registration, binding and installation process as appropriate and easy in terms of usability. The few participants who found it too complex or had other concerns described themselves as *"untalented"* computer users who often asked others for help. All participants described the process of sending encrypted messages and reading encrypted messages as *"normal"* and non-disruptive and most would use FBMCrypt to send all their private Facebook messages if their friends were using it as well.

All but one participant noticed the visual security indicators and most of the participants connected them to *"message security"*. An interesting finding during the interviews was that the displayed ciphertext was perceived as a trustworthy indicator for functioning encryption while the green and red borders were not. This aspect should be explored in more detail in further studies. A second interesting finding is that while the participants confirmed good usability attributes, the problem of establishing trust was described as something FBMCrypt itself could not provide. Instead, third parties were described as sources of information and trust. Some participants seemed to expect more overhead when encrypting a message. While Whitten and Tygar [201] as well as Garfinkel and Miller [92] showed that too complex a system results in rejection, the question of whether an appropriate amount of overhead could improve the perceived privacy and hence increase acceptance is an interesting one.

3.6 Limitations

This work has the following limitations. Precision: due to the within-subjects design of our lab study, carry-over and fatigue effects could have affected the study results. While a brief between-subject analysis based on the latin square setup did not show any worrying trends, a larger dedicated between-subjects study would be needed to rule out these effects.

Generalizability: Participants were all university students, selected for their frequent use of Facebook and their desire for Facebook message privacy. We believe the two selection criteria are valid, since this is the target group of our Facebook encryption mechanism. However, future studies of participants outside the university's demographic is of course desirable. Additionally, extending the sample to include non-privacy-aware users could also yield interesting insights into why people do or do not wish to protect their messages and how technology affects this.

Realism: The participants were restricted to using the computer provided for them during the study and using dummy Facebook and email accounts. Furthermore, only the first-time user experience was studied; we did not examine daily usage behaviour. Long-term studies using real Facebook accounts would address this.

3.7 Summary

In this work, we presented several user studies concerning conversation security on Facebook. In an initial screening study with 514 participants, we showed that within our student population, there is a desire to protect Facebook conversations. We identified two key design features of existing solutions: automatic or manual key-management and encryption. In a laboratory study with 96 participants, we tested the four combinations of these features using mockups and found highly significant preferences for automatic key-management and automatic encryption. Furthermore, participants who were worried about forgetting their password or losing access to their previous conversations stated that they would not use a mechanism without password recovery. Even though the automatic mechanisms had a higher acceptance rate, we also found that the two more complicated encryption mechanisms generally made the participants feel better protected.

As a result of our findings in the user studies, we designed and implemented an encryption mechanism for Facebook conversations. Several key design decisions were made to provide good usability. A service-based approach was chosen, providing confidentiality and integrity with automatic key management and recovery instead of burdening the user with complex cryptographic details. Security/usability trade-offs in this work were made considering the context of the Web 2.0 and Online Social Networks. For cases where these trade-offs are acceptable, our solution offers better usability than the email encryption systems tested in previous Johnny studies: All our study participants successfully encrypted their Facebook conversations without making any mistakes.

The interviews conducted during the final study revealed that usability alone is

not a sufficient incentive for accepting a mechanism for message security on Facebook. Many interviewees stated that actually seeing the mechanism do something – displaying ciphertext for example – heightened their perceived protection. However, we also found considerable distrust of security software in general. Participants stated that they would need to be convinced of the correctness by friends or trusted third parties, such as computer magazines before entrusting sensitive information to a message security mechanism. While this last statement was made after using the presented mechanism for Facebook conversations encryption, the interviewees often stated that this was a general attitude they had towards unknown security software.

This chapter gave new and promising insights into the motivation for end users to adopt an end-to-end encryption mechanism. While it illustrates the potential of well established usability, its limitations are made clear. A motivated and risk-aware user can chose to use a secure mechanism to protect information. However, security incidents and data breaches in the past [10] showed that even end users with a very strong interest in their information's security and privacy can easily become vulnerable by the actions of inexperienced or careless system administrators. The next chapter illustrates this connection: To make sure that information on the web is protected, end users can take care to use only secure HTTPS connections. Modern browsers indicate secure HTTPS connections by showing passive security indicators or extended validation information in the address bar. Looking out for these indicators is the only action end users can take to make sure their connections are secure. However, whenever administrators of websites decide to only provide plain, insecure HTTP connections or deploy invalid HTTPS certificates, end users' options are very limited. In those cases, they can only choose not to visit an insecure website.

To take this circumstance into account, the next chapter describes a study with administrators of websites. In the next chapter, I investigate their motivation to operate secure HTTPS enabled webservers and propose measures to improve the status quo.

[10]cf. `http://arstechnica.com/security/2016/01/time-warner-and-linode-report-possible-password-breaches/` – last access 13.04.2016

4 Administrators: Configuring HTTPS Webservers

Disclaimer: *The contents of this chapter were previously published as part of the paper "Why Eve and Mallory (Also) Love Webmasters: A Study on the Root Causes of SSL Misconfigurations" presented at* 9th *ACM Symposium on Information, Computer and Communications Security (ASIACCS) in 2014 [73] together with co-authors Yasemin Acar, Henning Perl and Matthew Smith. As this work was conducted with my co-authors as a team, this chapter will use the academic "we" to mirror this fact. The idea and initial concept for this work came from myself. The user-study with administrators who operated misconfigured HTTPS enabled webservers was designed and conducted by me. However, my co-authors contributed in different ways. Analyzing the study results was joint work with Yasemin Acar. Before compiling the paper for publication, Yasemin Acar, Henning Perl, Matthew Smith and I jointly discussed the study's implications.*

4.1 Motivation

For the authentication of a server during a TLS handshake, clients perform multiple validation steps to check whether the server's X.509 certificate is trustworthy or should better be rejected. Self-signed and expired certificates, certificates that were signed by an unknown CA, certificates that are not delivered with a complete issuer chain as well as certificates issued for the wrong hostname result in rejection. Whenever there is a problem with X.509 certificate validation, modern webbrowsers generate warning messages so users can decide how to proceed with the (possibly) critical X.509 certificate in question.

Previous research has shown that users tend to click-through these warning messages without paying attention to them, since supposedly most of the TLS warning messages users see when surfing the web are false positives, e. g. resulting from misconfigurations on the server side and are not real Man-In-The-Middle (MITM) attacks [4, 188]. Further research revealed that misconfigurations of HTTPS-enabled webservers are a widespread issue. The EFF conducted an internet-wide scan of all public IPv4 addresses on port 443 and collected the respective X.509 certificates[1]. Thus, they illustrated the state of X.509 certificates and presented details of X.509 certificates in use concerning the questions which CAs are in use, which certificates are self-signed, which certificates are expired etc., all of them essential for the TLS-handshake. Since then, multiple projects crawled the public part of the Internet for HTTPS certificates and analyzed different aspects of their deployment in the wild. Holz et al. [113], Akhawe and the ICSI Certificate Notary [3] collected X.509 certificates either actively or passively and concluded certain properties of the current CA-based TLS infrastructure:

While previous research provides valuable insights into the current TLS ecosystem, their focus is solely on technical aspects of TLS configurations or on the behavior of webbrowser users confronted with TLS warning messages but leaves out the following interesting questions: (1) Why are HTTPS-enabled websites operated with non-validating X.509 certificates at all? (2) How many misconfigured websites

[1] cf. `https://www.eff.org/observatory` – last access 13.04.2016

are frequently visited with webbrowsers and hence throw TLS warning messages? (3) For how many users do TLS warning messages occur unexpectedly? Based on the knowledge that a non-negligible percentage of TLS handshakes fail and that a large percentage of warning messages is dismissed by users [4], to the best of our knowledge we conducted the first qualitative study with website administrators to investigate the root causes for X.509 misconfiguration that cause browser warning messages.

We collected 755 study results to assess the motivation for the use of non-validating X.509 certificates on the web. We were interested in the reasons and motivation for administrators to operate non-validating X.509 certificates, how these administrators assess the operation of their non-validating certificates and the number of affected users and the protected data types. Additionally, we were interested in suggestions to improve the usability of certificate configuration.
Our findings suggest that a remarkable number of websites that operate non-validating X.509 certificates either do so intentionally or are not actively in use and hence do not trigger warning messages at all. However, we also find that many administrators misconfigure their HTTPS webservers due to the high complexity of TLS configuration options or due to a misunderstanding of the security features of TLS.

Our contributions can be summarized as follows: (1) We conduct the first user-study with administrators of HTTPS-enabled websites to identify the root causes for TLS warning messages in modern webbrowsers. (2) We find that a large amount of non-validating certificates is meant to be that way and clicking through them can be classified as deliberate. (3) We find that mainly websites with a manageable user count throw certificate validation errors. In many of these cases the users were previously informed, have probably been helped with the installation of the respective CA or trust the certificate and thus are not shown a warning message when browsing the site. (4) We report that a remarkable amount of websites employing non-validating certificates as can be found by crawlers are not meant to be actively used, are only ever accessed by crawlers and thus do not trigger real world warning messages for users. (5) We find that a substantial number of administrators are overwhelmed by the complexity of TLS and the configuration parameters offered by HTTPS webservers. (6) We provide a list of suggestions to improve the usability of X.509 certificate configuration on webservers given by the administrators.

4.2 Background

Interesting related work to TLS and its security and usability challenges can be found in Section 2.3.3

Webserver Configuration

Most webservers provide the option to serve content via HTTP and HTTPS. The Apache HTTP Server Project is a widely deployed webserver that supports both

HTTP and HTTPS. All configuration options such as vHosts, SNI and TLS parameters are placed in one or multiple configuration files. To enable TLS for an Apache webserver host, `SSLEngine on` has to be added to the configuration file. If no X.509 certificate is given via the `SSLCertificateFile` and `SSLCertificateKeyFile` directives, the default self-signed snakeoil X.509 certificate is used. SNI is configured by adding the previous `SSLCertificate` directives to a `VirtualHost` environment. The given X.509 certificate is then mapped to the `ServerName` present in the VirtualHost environment and is sent in the ServerHello message whenever the client's SNI values match the VirtualHost's ServerName value.

Listing 4.1: Apache configuration for an HTTPS-enabled website

```
<VirtualHost 192.168.0.1:443>
    ...
    SSLEngine               on
    SSLCertificateFile      ssl.crt
    SSLCertificateKeyFile   ssl.key
    ServerName      www.example.org
    ...
</VirtualHost>
```

While the basic scenario for X.509 certificate configuration is straightforward, more complex environments can easily lead to misconfigurations on the server side and TLS warning messages for the websites' users. In case the certificate in use was signed by an intermediate CA, the administrator needs to set the `SSLCertificate-ChainFile` parameter. Otherwise browsers will throw a warning message. In case the administrator's browser previously cached the intermediate CA in use, no warning message is shown, probably leaving the administrator unaware that other users will be shown a warning message. Another more complex scenario is the use of virtual hosts. Whenever multiple virtual hosts are operated on the same server, the correct certificate must be configured for each virtual host which can easily become a complex task and lead to false positive warning messages.

4.3 Administrator Study

While previous research either focused on a technological analysis of the deployed X.509 certificates in the wild or evaluated the users' behaviour when faced with a TLS warning message, our work incorporates the third important pillar in the TLS infrastructure: the administrators of HTTPS-enabled websites. Knowing the technical reasons why TLS handshakes fail and produce warning messages and how users react to those warning messages are important aspects. However, to achieve a better understanding of the whole picture, we conduct the first study with administrators to assess the root causes why administrators operate non-validating X.509 certificates. We think this will help both the research community as well as practitioners to work out more usable solutions in the future, to understand the reaction of users when faced with a TLS warning message and to reduce the overall occurrence of warning messages. To understand the motives of administrators to operate non-validating X.509 certificates we conducted an online survey with 755

Error Type	#Certificates	
Valid	3,876,497	86.38%
Self-Signed	89,981	2.0%
Expired	309,350	6.89%
Hostname Mismatch	146,941	3.27%
Unknown Issuer	64,694	1.44%

Table 4.1: Distribution of certificate validation error types.

administrators. The following section will give details on the methodology and will present and discuss the results of our study.

Methodology

To find websites that operate non-validating X.509 certificates, we gathered certificates deployed in the wild in a first step. We applied a technique different from previous work to collect X.509 certificates from websites: We used the body of certificates Google's webcrawler collected over a period of 12 months. The webcrawler collected X.509 certificates for 55,675,334 ($\sim$ 55.7 million) different hosts, resulting in a body of 4,487,463 X.509 certificates. This certificate body overcomes two essential problems common to other approaches reported in literature: (1) Actively crawling X.509 certificates for the complete IPv4 space such as Holz et al.[113] and the EFF TLS observatory resulted in a comprehensive map of IPv4 addresses and corresponding X.509 certificates – in this case one cannot deduce for which hostname the certificate was configured, hence post-validation does not allow for hostname verification. (2) Passively recording X.509 certificates similar to Akhawe et al. [3] only collects X.509 certificates for the websites their users visit – although they collected both X.509 certificates and Server Name Indication (SNI) [19] values for the corresponding TLS handshakes, they might have missed an essential part of the HTTPS-enabled part of the Internet. The X.509 certificate body of Google's webcrawler provides both, an encompassing list of X.509 certificates and their corresponding hostnames of the publicly available part of the Internet and the possibility to perform all three steps of X.509 certificate validation in postprocessing steps: (1) CA signature validation, (2) expiration checks and (3) hostname verification. We used the webcrawler's certificate body and performed the following steps to select candidates for our study: Firstly, we re-validated all X.509 certificates, using the NSS library as proposed by Akhawe et al. [3] which gave us the following results:

Altogether, our re-validation left us with 610,966 X.509 certificates that generate warning messages when users visit the corresponding websites. We picked a random sample of 50,000 of all failed X.509 certificates[2] and subsequently re-visited all websites of our 50,000 certificate sample to learn the current TLS configuration status of the corresponding webserver. This left us with 46,934 X.509 certificates and their corresponding webservers. In this remaining sample, the certificate validation error distribution was as follows:

[2] We conservatively estimated a success rate of reaching 10% of the administrators, and another 10% response rate to the study, which would have provided us with 500 answers

Error Type	#Certificates	
Self-signed	7,016	14.95%
Expired	22,952	48.9%
Hostname Mismatch	12,287	26.18%
Unknown Issuer	4,679	9.97%

Table 4.2: Distribution of certificate validation error types in the remaining sample.

We decided to get in contact with all of the affected administrators. Therefore we started by extracting email addresses from the collected X.509 certificates. Whenever we found an email address pointing to a Certificate Authority's or a webhosting provider's info address, we ignored it. For all other email addresses, we ran DNS queries for MX entries for the email's domain. In case of a positive response, we stored the email address for contacting the administrator later. We checked those email addresses for duplicates, so we would only contact any given administrator once, to not unnecessarily bother them. In order to contact the administrators for whom we did not find email addresses embedded in the certificate, we decided to send an email to `webmaster@domain.com`, as the administrator email address is specified in RFC2142 [45] as a recommended address for any domain. To avoid sending multiple emails to one recipient and to improve the chances of actually reaching the person responsible for the certificate configuration in question, we did not target the `postmaster@domain.com` or `abuse@domain.com` email addresses. To avoid triggering spam warnings, we sent out the emails very slowly at a rate of 2,000 emails per day. Altogether, we sent 46,145 emails to email addresses either embedded in an X.509 certificate or to the domain's corresponding administrator email address. We sent 40,480 emails to `webmaster@domain.com` and 5,664 to embedded addresses. 37,596 of those could not be delivered, leaving us with 8,549 successfully delivered emails (cf. Appendix B, Page 197 for the email we sent). We received 755 complete responses to our survey, a response rate of 8.83%.

We decided on a set of questions that would take only 5-7 minutes to answer, including two free text questions why the administrators were using exactly this X.509 certificate on their website and the free text prompt to report problems with the configuration and wishes to make configurations for HTTPS more usable. We were mainly interested in the following aspects:
(1) **Reasons and Use Cases for employing HTTPS:** We were interested in how the website was primarily accessed, how many users were visiting it and in which context (e.g. commercial, private etc.) the website was mostly used.
(2) **Technical Knowledge concerning TLS:** We asked several questions to asses how much the administrators knew about TLS and if they had set it up themselves; we asked for estimations for the pricing of X.509 certificates and problems they had with TLS.
(3) **Risk Assessment Concerning Misconfigured TLS:** We asked how important TLS was for their website and how strong the risk for users was due to the non-validating certificate.
(4) **Complaints, Wishes and Suggestions for TLS:** In the end, we asked them to fill in a free text about if they had problems with configuring the certificate for

their webserver, also asking them for complaints and ideas to "make things better".

Ethics

The study was conducted in Germany and thus was not required to pass an IRB review. Nevertheless, our studies complied with the strict German privacy regulations. We discussed our study design and goal with the Privacy Officer. The purpose of making contact with the affected administrators was two-fold: (1) we intended to inform administrators of the misconfiguration of their website and (2) kindly asked them to support our research. However, administrators would benefit from our email without participating in the study.

We were aware that sending emails to all candidates at once could cause resentment in the recipients. To reduce negative side-effects, we specifically contacted the administrator of the website by sending an email to the `webmaster@domain.com` address that is specifically intended for questions and comments concerning technical problems, as stated in RFC2142 [45]. However, we felt we were offering the administrators valuable information, as we only contacted those who operated an invalid certificate and pointed this out to them. We did not follow any commercial goals and went out of our way to ensure that we would send only one email to each administrator. Hypothetically, any Internet user browsing any of those websites would have encountered a warning message, which would have asked them to contact the administrator. Additionally, we kindly asked them to fill out a survey and explained our research interest to them. Altogether, when we decided whether to contact the administrators, we needed to weigh the benefits for the individual administrators, their websites' users and – following our survey – the whole Internet community against the risk of being seen as spam. Our results confirm our estimation of the situation: most administrators reacted in a grateful or at least friendly way, some nicely explained why we saw the invalid certificate or thanked us for alarming them to the non-validating certificate and some of them said they wanted to fix the TLS configuration of their websites immediately.

Only three administrators complained about our email (cf. Section 4.3). Since we did gather valuable results from the survey, and received mostly positive feedback, we argue that even though we were not directly prompted to send out the emails, we were not at all generating spam.

Handling Complaints

When we planned to send several thousand emails to administrators of misconfigured HTTPS-enabled webservers, we were aiming for a fairly low number of complaints. Therefore we set up a website which explained the intention of the email and described our study. The link to this website was attached to every email we sent out. We also offered an email address for further contact. Overall we got mostly positive feedback from the site administrators we contacted for participating in the study. Only 19 of the candidates complained about receiving our email at the first step. 16 of those 19 wanted to find out the intention of our data collection and if the study

and research group was real. After we contacted them, answered all their questions and explained the purpose of our study to them, all of them responded very friendly and agreed to fill out the survey. Only three administrators explained that they had no interest in participating in the study and demanded being added to a blacklist to prevent any further study invitations. Of course, we complied with their wish.

Study Results

Of the 755 administrators, 154 (20.4%) operated websites with an expired certificate, for 250 (33.1%) websites hostname validation failed (13 were also expired), 160 (21.2%) websites used an X.509 certificate issued by a CA not included in the Mozilla truststore and 191 (25.3%) websites operated self-signed certificates.

Reasons and Use Cases for employing HTTPS

The primary access method was said to be via browsers in 681 cases, 15 by apps, 15 by embedded systems, and 44 stated they did not know. 319 administrators estimated they had less than one hundred visitors per month, 165 estimated between one hundred and thousand, 95 between thousand an ten thousand, 66 between ten thousand and a hundred thousand, 19 between a hundred thousand and a million, 5 more than a million. We asked them to rate a valid certificate's value. 242 stated a valid X.509 certificate is worth 0$, 253 stated an X.509 certificate is worth between 1$ and 20$, 93 stated the worth between 20$ and 50$, 80 stated the worth between 50$ and 100$, 40 stated it between 101$ and 500$, 13 stated it between 501$ and 1000$ and 34 stated the worth of a valid X.509 certificate more than 1000$. Of the 134 administrators who offered information about their websites' users, 84 (62.7%) said it was used only by themselves (primarily for administrative purposes), 11 (8.2%) said it was mostly used by friends and 39 (29.1%) said it was used by their company and colleagues. An important question of interest was why exactly the websites operated a non-validating X.509 certificate. Of the 495 administrators who gave information as to why the certificate was configured in a way that would throw a warning message, 330 said they had configured it in such a way on purpose. W713, who operated a self-signed certificate, stated: *"The site is a development system not accessed by customers or the public and the warning message "issue" is known internally."*, while W49 stated: *"The X.509 certificate is used for access to sensitive parts of the site. It is only being used by skilled operators, i. e. people who are able to check the fingerprint of the certificate to determine its authenticity and then store it for subsequent uses."* W23 on the other hand mentioned: *"Using TLS with a commercial CA issued certificate that is not under total control by myself is inherently insecure, since every CA owner can hijack the security and all providers that acquired an intermediate CA certificate can do so. And in the last years we have seen how weak some CAs are protected against cybercriminals. So it's much more secure for users to accept a certificate that was signed by my own CA once and get cautious when it changes."* Another prominent statement came from W31: *"Our users are explicitly required to provision the CACert.org root CA before visiting the website. The website generates no warning then."* As can be seen, a fair amount of

Error Type	Deliberate	Misconfiguration	Not Actively Used
Self-Signed	90	45	20
Expired	74	38	16
Hostname Mismatch	82	50	51
Unknown Issuer	84	32	14
Total	330	165	101

Table 4.3: The distribution of different error types between administrators that deliberately use non-validating X.509 certificates, made a mistake while configuring their servers or stated that their HTTPS-enabled webserver is not actively used any more.

administrators made a fairly informed decision not to employ a validating certificate and explained why they did what they did. *Site operates a self-signed certificate: "I just wanted to try HTTPS on our website and the used X.509 certificate is the default Apache provider template." (A595) Site operates a CACert certificate: "Our users are explicitly required to provision the CACert.org root CA before visiting the website. The website generates no warning then." (A31)* While 495 administrators stated they use the questionable certificate deliberately, 165 administrators said they had accidentally misconfigured their X.509 certificate. Again, the reasons for employing a non-validating certificate are manifold. W218 for example stated, *"I am the administrator of a website in the medical domain that must be HIPAA compliant and the HIPAA guidelines require HTTPS for websites. Since we did not want to spend money on a commercial X.509 certificate we decided to use a self-signed certificate."*, while W284 stated *"Actually you see the warning message because the certificate was issued for xxx.com and not www.xxx.com. If you click through the warning message you will be redirected to the correct website and will see no warning message at all."*. W98 stated *"I'm using one certificate for many sites (my server did not support SNI until a recent update), so I had to list every one of them in the alternative domain name. Since it is tiresome to add an additional entry for every one of them to account for the the correct subdomain, I did not bother [...] But your survey brought up to my attention this cases and I will fix the issue immediately"*
Site operates an expired certificate: "This domain is an old domain only used as a redirect to our new domain. This domain was last used over five years ago. We deemed it was no longer worth the money to purchase an X.509 certificate." (A190)
The *deliberate setup* group self-reported a mean TLS technical knowledge of 4.08, while the *misconfiguration* group reported a mean TLS technical knowledge of 3.80. The deliberate group rated their data sensitivity to be a mean of 2.48 out of 5, while the misconfiguration group rated it to be a 2.43. 101 of the participants stated that their websites were not actively in use or said that they were sure that there were no hyperlinks pointing to their website and hence no browser warning messages would ever be thrown.

Technical Knowledge Concerning TLS

We asked the administrators who had set up the X.509 certificate for their HTTPS server. 613 stated they had set it up themselves, 63 certificates were set up by a coworker, 12 by a retired coworker, 68 by their service provider and 11 did not know who set up the certificate. We asked the participants to self-report their technical knowledge of TLS on a 5-point likert-scale between *very low* and *very high*. 12 self-reported their TLS knowledge as *very low*, while 236 said their technical knowledge of TLS was *very high*. In the mean they rated it as a 3.96. Many administrators were uninformed about the pricing of X.509 certificates and strongly overestimated the actual costs. We asked how much they think a valid X.509 certificate costs [3]. 181 administrators estimated a valid X.509 certificate to cost 0$, 114 said it would cost between 1$ and 20$, 145 said it would cost between 21$ and 50$, 156 said it would cost between 51$ and 100$, 131 said it would cost between 101$ and 500$, 24 said it would cost between 501$ and 1000$ and 4 said it would cost more than 1000$. A total of 87 (11.5%) administrators did not know their configuration could lead to browser warning messages prior to our survey (those who did know reported a mean technical TLS knowledge to be 4.02, those who did not reported their mean technical TLS knowledge to be 3.50). One interesting finding was that six survey participants stated they do not need the features provided by CA-issued X.509 certificates. They argued that for their use cases they did not need the authenticity features CA-issued certificates provide but only rely on “strong encryption” to transport sensitive data. These statements demonstrate a lack of understanding of the TLS security features. Although they rely on strong encryption, they oversee the fact that without properly verifying the identity of the server, no secure communication channel can be established in a reliable way since a MITM attacker could easily exchange the original with a malicious certificate.

Risk Assessment Concerning Misconfigured TLS

We asked the participants to rate the importance of HTTPS for the operation of their website on a 5-point likert-scale from *not important at all* to *very important*. 217 said that HTTPS is not important at all, while 190 stated that it is very important for their websites’ users. In the mean they rated it as a 3.00. We also asked them to rate the sensitivity of the data their website serves via HTTPS on a 5-point likert-scale ranking from 1 as *not sensitive at all* to 5 as *very sensitive*. 252 rated the sensitivity of their data as not sensitive at all, while 81 rated it as very sensitive. In the mean they rated it as a 2.53. We also had them rate the risk the non-validating X.509 certificate they are using poses to their users. On a 5-point likert-scale they could choose from *very low* to *very high*. 524 rated the risk as very low, while 23 said the risk for their users was very high. In the mean they rated it as a 1.55 Of the 755 respondents, 612 stated their users never complained about the occurring warning message, 7 administrators reported they receive complaints at least once a week, 12 received complaints on a monthly basis, 26 on a yearly basis, 77 receive

[3] We asked both questions – to rate a valid certificate’s value and how much our participants think a certificate costs – to measure the perceived value versus the assumed monetary cost.

complaints less often and 21 could not remember how often they receive complaints. We had been looking for interesting correlations between certificate error types and certain self-reported values. However, except for the results described above, we did not find a statistically significant correlation that would have helped us predict what kind of error would occur because of which characteristic. This underlines the importance of increasing the usability of X.509 certificate configuration and deployment in general, as well as building in more failsafe mechanisms and generally taking the weight of correct and secure X.509 certificate configuration and deployment away from administrators.
At the end of the study, we asked the administrators to describe problems they encountered with setting up TLS for their website and suggestions they had to *make X.509 certificate configuration more usable*. We present their concerns and suggestions in what we call the *Admins' Wishlist*.

Admins' Wishlist

We asked our participants to describe improvements they would like to add to make X.509 certificate configuration for HTTPS webservers easier and what they think is missing in the current system. In the following section we analyze their statements and describe what most participants find lacking in the current system. Of the 755 responding administrators, 87 offered suggestions, some of them more than one. Their suggestions can be categorized into six different groups:

Lowering The Price: 13 of the participants mentioned that the current price range for X.509 certificates that do not throw warning messages in browsers is not adequate. They find that paying a high amount of money for such a low cost task such as digitally signing an X.509 certificate is not fair and they would like to see a change in the current pricing policy of commercial Certificate Authorities. They criticize that the current CA infrastructure *"is a money printing machine without providing strong security for both service providers and their users"* (W29). Nine of them asked for a CA that issues free certificates that are accepted by popular browsers. Four participants complained that they have to configure X.509 certificates for multiple subdomains and that current wildcard certificates are too expensive. They wished to get access to cheaper wildcard certificates to reduce the number of false positive warnings on their websites.

Allowing CACert: 45 websites operated an X.509 certificate issued by CACert[4]. 10 of these proposed to add the CACert root CA to all popular browsers to provide an alternative to the commercial CAs issuing trusted certificates. The motivation to use a CACert certificate was two-minded: 28 of the administrators preferred CACert certificates since they did not want to support the commercial CAs and are of the opinion that basic encryption mechanisms as provided by TLS should be accessible by everyone for free. The remaining (17) did not trust the centralized trust model of commercial CAs after the breaches of DigiNotar and Commodo. They argued that the CACert's web of trust model provides more security and better protection against Certificate Authority compromise attacks.

[4]cf. `http://www.cacert.org` – last access 13.04.2016

Better Support for Non-Validating Certificates: 15 participants complained that they were forced to use certificates issued by commercial CAs to avoid TLS warning messages. They can be categorized into three different groups: Seven participants would like to change the current trust model. While two did not describe their idea of a different trust model, three would prefer a trust-on-first-use-based model such as known from the Secure Shell [205] which would allow them to use a certificate of their choice. Two other responders would prefer the TACK trust model proposed by Moxie Marlinspike[5] since they explicitly did not trust commercial CAs. Five participants would like to have an easy way to use self-signed certificates without giving concrete ideas of how such a system could work and four participants wanted to have an easier-to-use mechanism to validate certificate fingerprints to be able to securely deploy self-signed certificates for their users. Three participants were using their own CA in an enterprise environment and criticized the complicated workflow of adding their custom CAs to their users' browsers.

Better Tool Support: Six survey participants suggested to improve the tool support to generate and configure X.509 certificates for webservers. They found the command line interface for the OpenSSL tool[6] too complicated and wished for better documentation. The TLS configuration options of popular webservers were also criticized. Particularly the configuration of virtual hosts was described as very complicated and error-prone and administrators generally requested a more easy to use mechanism to configure X.509 certificates for multiple hostnames on a single IP address.

Auto-Update Reminder: Eight survey participants who used an already expired certificate were not aware of that fact before we contacted them. They criticized the fact that they would not receive an automatic message when their certificate expired and would like to have a service that keeps an eye on the expiration date of their certificate: *"Ideally an automatic message would be sent out to not miss the date to re-new a server's certificate"* (W643).

4.4 Discussion

Our study reveals new findings and helps to better understand previous work in the field. While Akhawe and Felt [4] desire a 0% click-through rate for TLS warning messages, in our study 330 of 755 website administrators stated that they deliberately operate non-validating X.509 certificates and that their users are informed of the warning message beforehand. In these cases, TLS warning messages are no unexpected security warnings, but can be seen as information dialogs that users expect and to which they react by clicking through the warning because their administrator told them to. For users who know how to verify the fingerprint of an X.509 certificate, deploying a non-validating X.509 certificate does not pose an extra security risk. However, manually verifying the fingerprint of an X.509 certificate is not trivial for the average Internet user and it might be even easier to trick the non-technical

[5] cf. `http://tack.io/draft.html` – last access 13.04.2016
[6] cf. `http://www.openssl.org/` – last access 13.04.2016

user into clicking-through an attacker's X.509 certificate: In situations where users expect a TLS warning message and are said to click-through by their administrator they have almost no chance to differentiate an attacker's X.509 certificate from the non-validating but benign X.509 certificate installed by the administrator.
Whenever websites with non-validating certificates are re-visited by users, and the users did not add the non-validating certificate or the browser-untrusted CA to their truststore, they will repeatedly click through the warning. Since Google Chrome does not open the change to its trust store in the warning menu, it is likely that users will click-through a warning message on every visit. This is one possible explanation for the huge difference in click-through rates as reported by Akhawe et al.[4]: They count every repeated click-through in Chrome, while they count click-throughs in Firefox only at the first visit to the respective website (which may or may not have occurred during the period of their data collection). Thus, our findings support the assumptions that Chrome's click-through rate is massively influenced by re-visits of websites that operate non-validating certificates.
We found that many administrators reported that their site was either not in use any more, or that the TLS version of the specific domain we encountered had never been meant to be accessible for users at all and had respectively never been hyperlinked anywhere on the Internet. This attests to a very important finding: Studies using datasets of TLS certificates that were accumulated by certificate crawlers are prone to massively overreport handshake failures and hence TLS warning messages in browsers not only by assuming a possibly not applicable set of trusted certificate issuers, but also by including unused websites.
We found that the administrators generally rated their knowledge about TLS to be rather high, which is surprising: We would have expected there to be two sets of administrators that strongly differ in their knowledge about TLS: Those who deliberately configured their webserver in a way that their X.509 certificate does not validate, and those who accidentally misconfigured their webserver. We would have expected the former group to self-report a remarkably higher technical knowledge than the latter group. While they did report a 4.08 versus a 3.80, compared to the difference in skill they demonstrated, this difference in self-reporting seems rather insignificant. We gained valuable insights from the free texts the administrators wrote about problems with TLS and improvement suggestions. Many of them wished for more simplicity: 165 had accidentally misconfigured TLS. Some wished for either a simpler interface to set up a webserver, others wanted an automatic renewal for expiring certificates.
330 administrators had configured their webservers in a non-validating way on purpose. 15 of them wished that there was a broadly-accepted alternative to commercial CAs; in general there were complaints about the pricing of CAs. This is a very interesting finding: 20 of the 85 administrators who suggested improvements requested a free alternative to paid CA certificates. Obviously these administrators were not aware of the fact that there are free alternatives[7] that provide free and trusted X.509 certificates, which demonstrates that there is not only the need for a better technical education but also for a broad and basic documentation, complete with examples

[7]cf. `https://letsencrypt.org/` – last access 13.04.2016

and links for administrators, who understandably do not call TLS their primary field of expertise. While there exists a fair number of alternatives and suggested improvements to the current TLS infrastructure [144, 200, 145, 135], none of them has been widely deployed yet. They each come with a collection of advantages and disadvantages over the current TLS system. As we could see in our study, many administrators are confused and overwhelmed by configuring X.509 certificates correctly. These findings reveal that any new TLS system not only has to improve security and usability issues for end users but also needs to focus configuration and deployment usability for administrators.

4.5 Limitations

Population: We contacted administrators from a random sample of 50,000 websites which operated non-validating X.509 certificates without considering the popularity of the given website. While this might have resulted in contacting many websites that are only rarely visited, this was of very much interest in the context of our study. Our results imply that X.509 certificate warnings occur more frequently on websites with low traffic which is often regarded as unproblematic by their administrators: they claim that their users are aware of the presence of a TLS warning message causative certificate.
Self-Selection Bias: All our participants were self-selected. They chose to fill out the survey, which could mean that more active administrators answered.
Bounced emails: We tried to reach administrators either by using the contact email address in the website's X.509 certificate or the `webmaster@domain.com` email address. 37,596 of all emails we sent were bounced. Hence, the majority of websites with non-validating X.509 certificates does neither follow best practices and nor provide an easy-to-find contact email address. It might be possible that those administrators have different reasons for using a non-validating X.509 certificate on their websites.
No incentive: We did not offer a monetary incentive to our study participants. A study by Callison [33] shows that volunteers offer good work. Our decision to focus on volunteers might have prevented administrators that expected some kind of monetary reward for answering a short research survey. Hence our results might not reflect their motives for operating non-validating X.509 certificates. While it is common-practice to pay participants of end user studies, we decided to not offer a financial incentive to the website administrators since we found it very hard to for several reasons. First of all, monetary incentives could increase the probability of being seen as spam. Secondly, it would have been impractical to pay them, since they would have had to trust us with some banking account information.
Underreporting: Some of our conclusions are drawn from answers which were given as free text. Thus we do not have data on all of our participants for several issues: Some did not report on whether their website is in use/is meant to be used at all, while others did. Not all of the users report on who the TLS connection is intended for. Therefore it is possible that we underreport the websites which are

out of use, as well as the websites which exist for administrators' use only.

4.6 Summary

We conducted the first study with administrators who operate non-validating X.509 certificates on their HTTPS-enabled websites to understand their motives. Therefore, we used the body of 4,487,463 certificates Google's webcrawler had collected over 12 months. We identified 610,966 non-validating certificates, chose a random sample of 50,000 of these, established if they were still operating and non-validating certificates, extracted email addresses for their administrators and emailed them. Of those emails, 8,549 were successfully delivered. Of these, 755 administrators who operated websites with non-validating X.509 certificates responded to our study.
101 said that their website was not meant to be accessible, and that actual users would not have encountered the certificate as the webcrawler did. We found that of the 495 who reported on this issue, 330 said that their use of a non-validating certificate was deliberate, while only 165 explained it with an accidental misconfiguration. 44 of the administrators (25% of those who had accidentally misconfigured their webserver) stated that they were confused about TLS configuration in general, strengthening the our assumption already made in previous research [81] that, while warning messages and user behavior are an important field of study, studies with IT professionals in general and administrators and developers in particular are an important and often neglected issue. We confirm findings from Akhawe et al., who state that Google Chrome users tend to click through warning messages more easily: This goes well with our finding that many administrators use non-validating certificates on purpose and inform their users about it. Clicking through these warnings will add the certificate to Firefox, while Google Chrome will show a new warning on each revisit.

This leads us to a point made by previous research. Habituation is a problem for users when confronted with warning messages, both in general, but also in the following specific scenario: Users who frequently visit websites where they expect warning messages do have a lowered risk perception for the respective websites. Those sites may seldomly be targets for Man-In-The-Middle attacks due to their low overall user count, but if an attack takes place, users who did not add the website's certificate or custom CA to their browser's truststore but simply click through the warning have no chance to differentiate between the expected warning and an actual attack.

This chapter gave new and promising insights into the motivation for administrators of webservers to (not) deploy valid TLS certificates, leaving the security and privacy of their end users at risk. However, the impact of administrators on an ecosystem's security and privacy strongly depends on the developers shaping the

ecosystem. For example, administrators of IT systems can enforce strong end user authentication by enforcing secure password policies, e. g. not allowing weak passwords that can easily be looked up in a dictionary. End users adhering to their administrators' password policies have different strategies at hand to manage passwords. One popular strategy is the use of a password manager. The overall system's security then not only depends on secure password policies defined by administrators and end users adhering these policies, but also strongly depends on the secure implementation of password manager software.

To investigate this circumstance and gain insights into the developers' role, the next chapter consists of two parts: The first part analyzes the security of mobile password manager software for the Android platform. The second part includes a study with developers of those apps to obtain a better understanding of challenges for developers during the implementation of secure password managers.

5 Developers: Implementing Password Managers

Disclaimer: *The contents of this chapter were previously published as part of the paper "Hey, You, Get Off of My Clipboard - On How Usability Trumps Security in Android Password Managers" presented at* 17th *International Conference on Financial Cryptography and Data Security (FC) in 2013 [80] together with co-authors Marian Harbach, Thomas Muders, Marten Oltrogge and Matthew Smith. As this work was conducted with my co-authors as a team, this chapter will use the academic "we" to mirror this fact. The idea and initial concept for this work came from myself. I developed the "pwsniff" attack, implemented the proof-of-concept exploit and analyzed all affected password managers. The interviews conducted with developers of vulnerable Android password manager apps were designed, conducted and analyzed by myself. However, my co-authors contributed in different ways. Marten Oltrogge assisted in implementing the "USecPassBoard" keyboard as a countermeasure for the "pwsniff" attack. Together with Marian Harbach, Thomas Muders and Matthew Smith, we jointly discussed the work's implications and compiled the paper for publication.*
Software described in this chapter is available at `https://zenodo.org/record/50527`.

5.1 Motivation

Text-based passwords are the most prominent authentication scheme in computer systems. Although researchers have been criticising this scheme as being hard to use in a secure way, it is still the most widely adopted system for authenticating users.

However, due to the bounded cognitive abilities and motivation of users, password re-use is commonplace (cf. Section 2.3.1). Password managers (PMs) aim to overcome this problem: they help the user to handle a large number of different passwords by storing them in encrypted form. To access the encrypted passwords, the user usually has to enter a single master secret that decrypts the password database. Password managers often include a password generator to simplify the creation of new unique and secure passwords. For convenience, login forms are pre-filled and account information for new websites is captured on the fly. Another prominent feature is the synchronisation of password databases between multiple devices.

While early password manager applications were limited to desktop computers and their browsers [168], current implementations offer mobile password manager apps, that can be synchronised over the cloud to ensure that the password database is available on all of the users' devices. In contrast to the smooth integration of password managers on desktop browsers, current mobile platforms lack such plugin facilities and therefore do not provide convenience features, such as automatic account creation or credential fill-in. While users of desktop password managers benefit from a smooth integration into browsers, password manager apps on mobile platforms offer less comfort. First, web browsers on smartphones and tablets often do not provide a plugin interface, that would allow for a smooth integration of password managers. Second, the existence of dedicated apps for many online services steadily increases the number of users that access online services through an app

instead of a website in a general-purpose web browser. This circumstance requires that mobile password managers have to be able to manage passwords not only for browsers but also for apps. Unfortunately, there is a fundamental problem with password manager apps on Android: The OS does not offer an API to integrate password managers with the browser or other apps. This has led to the adoption of a highly insecure practice to overcome this weakness: Password managers use the OS clipboard to transfer credentials from a password manager app to the browser or other apps. This method effectively broadcasts credentials to all apps installed on the smartphone.

5.2 Background

While early password managers were simply a username/password database embedded in desktop computers' browsers, the features of modern password managers are much more extensive and can be categorised as follows:

Browser-Embedded PM

Most web browsers, such as Google Chrome, Microsoft Internet Explorer, Mozilla Firefox or Apple's Safari, include an embedded password management feature. In case a user logs into a website for the first time, the PM inquires whether the login credentials should be saved to ease future logins by automatically filling the username and password into the login form. These embedded password managers traditionally store credentials locally on the user's computer, but are increasingly syncing them between multiple devices using proprietary Cloud services. Some browsers do not encrypt credentials stored locally or require the user to set a master secret to enable encryption. For example, Chrome uses the Google Account password to encrypt the synced password database by default, but offers to use a dedicated secret as an advanced feature.

Browser Plugins

The majority of modern desktop web browsers provide an API to extend their functionality by allowing the user to install third-party plugins or extensions. Many password managers are hence available as browser plugins. KeePass[1], 1Password[2] and Lastpass[3] are prominent examples of plugin-based password managers. They encrypt passwords and protect them with a master secret. These third-party password managers often provide further functionality and act as encrypted storage for more than just usernames and passwords: other sensitive information such as credit card numbers, online banking information or secret notes can be stored as well.

[1] cf. `http://keepass.info/` – last access 13.04.2016
[2] cf. `https://agilebits.com/onepassword` – last access 13.04.2016
[3] cf. `https://lastpass.com` – last access 13.04.2016

Password Managers on Desktops

Password managers on desktop computers are generally well integrated into the users' everyday Internet-facing software, such as browsers and email clients. Regardless of whether an embedded password manager or a third-party plugin is used, when the user accesses an online account for the first time or creates a new account, the password manager automatically comes into play and offers the user to securely store the new account information. The plugin APIs of modern browsers offer a very comfortable integration of password managers into the users' workflows. When a user visits a website that requires authentication, a password manager typically auto-fills the username and password and might even automatically submit the login form.

Early password managers for desktop computers (e. g. [168]) assumed a single device environment. Nowadays users often work with multiple devices such as desktops, notebooks, smartphones and tablet PCs, which makes it necessary to synchronise password databases between multiple devices to have credentials available whenever needed. For this reason, some password managers offer to sync databases between multiple devices by storing credentials in the Cloud or by putting the database on USB drives. A popular way to sync password databases is the Dropbox service. The encrypted password database is stored in the user's Dropbox account and can be accessed from all the user's devices. 1Password for example maintains an encrypted database for sensitive information and allows users to store and share the database via their Dropbox account.

Password Managers on Mobile Devices

While password managers in desktop environments are well integrated into browsers and users' workflows, the situation for third-party password managers on mobile platforms is different. Neither of the major mobile platforms (Android, iOS and Windows Mobile) nor mobile browsers provide a plugin API comparable to desktop computers. Additionally, the paradigm shift away from the browser as a generic tool to surf the Internet towards the "there is an app for everything" approach makes integrating PMs into mobile ecosystems even harder.

API limitations and the requirement to support arbitrary apps creates a different usage pattern for mobile PMs. Instead of storing new account information automatically and auto-filling authentication forms, the workflows of mobile PMs typically consist of the following steps:

1. The user has to switch to the PM app,

2. then needs to find the appropriate username/password tuple from a list of stored credentials,

3. copies the password to the clipboard,

4. switches back to the app that requires authentication, and

5. finally pastes the password into the corresponding text field before submitting the login form.

6. In case the user does not remember the username for a given service, these steps (except step 2) are repeated for the username as well.

Although this workflow's usability is far from optimal, it is the best mobile password managers can provide so far. To understand why users nevertheless use password managers on mobile devices, we analyzed 2,000 user reviews in Google's Play Market. To this end, we downloaded user reviews, manually extracted factors that motivate users to use PMs on their Android device and identified the following reasons to be substantial:

Protection: Users feel that embedded PMs do not store the passwords in a way they believe to be secure (e. g. some users were angry that Android's stock browser does not encrypt stored usernames and passwords).

Confidentiality Users do not trust embedded PMs in keeping their data confidential (e. g. users were afraid that their credentials could be sent to Google).

Features: Embedded PMs are usually limited to usernames/passwords. Users often want to store other confidential data, such as banking information.

Availability: Embedded PMs are usually limited to a single browser. Since many users need access to their information on multiple devices and browsers, a vendor-independent PM is preferred.

Related Work

Gasti and Kasper [94] analyze the security of 13 popular password manager databases and their storage formats. They find even popular password managers such as Google Chrome store password databases in clear text providing no security against attacks. However, even password managers such as Mozilla Firefox and Microsoft's Internet Explorer that use a master password to encrypt password databases are vulnerable against weak attackers. Overall they find that only one of the password managers they analyzed was secure against their attacks.

Li et al. [138] investigated the security of web-based password managers that run in browsers as plugins and store the password database remotely on the web. They evaluate the security of five different web-based password managers and find that four of them have sever security vulnerabilities that allow an attacker to learn users' credentials. The root causes are mainly developers misunderstanding the concepts of web security in addition to vulnerabilities such as CSRF and XSS.

Silver et al. [181] study the security of the most popular password manager software and their policies on automatically filling in passwords on websites. In addition to web-based password managers, they also investigate mobile password managers and standalone apps. Their work on mobile password managers extends our work in this chapter. They find that autofill policies are hard to implement correctly and can therefore lead to serious consequences allowing a network attacker to sniff credentials without any user interaction.

5.3 Password Sniffing on Android

As illustrated in Section 5.2, the workflow of mobile password managers requires the user to copy account credentials to the clipboard before switching to the target app and pasting them before actually logging in. There are some problems with this practice: On Android, writing data to or reading data from the clipboard does not require any permission. Therefore, every app currently running on an Android device can read the items stored in the clipboard at any time. To make matters worse for password managers, the Android SDK provides the `android.content.ClipboardManager.OnPrimaryClipChangedListener` interface, which defines a listener callback that is invoked each time the primary item on the clipboard changes. This can be used by malicious apps to harvest passwords as they are passed through the clipboard. As a proof of concept we implemented a password sniffer named *PWSniff* using this mechanism. PWSniff runs as a background service and does not require any Android permission to work properly.

As long as no changes to the clipboard occur, the background service idles and therefore does not consume any CPU cycles. Directly after a new item is copied to the clipboard, the listener callback is invoked by Android and the idling PWSniff background service is notified and then reads the primary item. Next, PWSniff determines the app which is currently in the foreground. This information can also be acquired without requesting any permission. We assume that the foreground app at the time of copying is the app from which a user copied data (cf. Section 5.2 step 1). In case this app is a known password manager, we assume that the primary clipboard item is either a service URL, a username or a password (cf. Section 5.2 step 3). Whether the app is a password manager can be determined based on the app's user ID, which is assigned at install time and can be mapped to the a unique app market ID. The third step in our attack is to wait for a foreground app switch by checking the current foreground app in a loop and waiting until the user brings another app to the foreground. In case we identified the primary clipboard item as possibly confidential data (no matter if it is a username or a password) copied from a password manager, the new foreground app is assumed to be the destination of the credentials-copy-operation (cf. Section 5.2 step 4).

Hence, by exploiting features of the Android SDK that require no special permissions in combination with a typical workflow in the context of using password manager apps on Android, it is easily possible to harvest (still potentially noisy) usernames and passwords from the world readable clipboard.

At this point, an attacker cannot be sure which item is the username and which the password. But, in many cases it is possible to differentiate between both items based on their structure. Usernames are often chosen to be easily memorable (e. g. an email address) while passwords, especially those which are managed with PM software, usually are more "cryptic". Even in cases where the username and password cannot be easily distinguished, an attacker could first try one combination of the sniffed items and in a second attempt the reversed order. In both cases, breaking into an account is straightforward.

Advanced Username Capture

The attack described above relies on the user copying and pasting both the username and the password. Since users might just type their usernames from memory or use browser or app autofill features to save this effort, it might become necessary to acquire the username through an alternative method. For this, PWSniff can be equipped with the GET_ACCOUNTS permission. The permission allows the app to see usernames that other apps handle on the smartphone and which are registered with the *AccountManager*[4]. This also includes all email addresses used on the device. Since many online services use email addresses as usernames, this list offers a good basis from which to guess usernames for many services.

The downside of this extension is that it involves the danger of a user becoming suspicious of the app's permissions, which are presented to the user at install time. However, Felt et al. [84] demonstrated that users pay little attention to the permissions of an app and mostly do not understand the permissions' meaning. While Felt et al's. results account for Android's permission system in general, an app's permissions are also grouped and classified based on their security relevance. In this respect, the GET_ACCOUNTS permission does not rank particularly high and thus is not often shown on the first page. To ensure that the GET_ACCOUNTS permission is not shown on the first page, an attacker only needs to request more than three popular permissions such as INTERNET, LOCATION and STORAGE which are used by many apps. The best composition of permissions to mask the GET_ACCOUNTS permission is outside the scope of this work.

Advanced Account Capture

In Section 5.3, we illustrated that an Android app which holds no special permissions is able to sniff online account credentials that are copied to the clipboard when working with any password manager on Android in most cases. It is also possible to learn from which app a value was copied to the clipboard and into which app the value was pasted. If the target app has a special purpose (e. g. the Skype app only logs into Skype), it is easy to guess to which online service the harvested credentials belong. However, in case the target app is a multi-purpose Internet client such as a web browser, finding the intended service is not quite as straightforward.

To learn for which account a password is used, an attacker can benefit from Android's ProcFS features. The ProcFS is an interface to the kernel and provides information about a device such as information about the CPU, memory and network details. On Linux-based systems such as Android, the ProcFS is usually mounted at `/proc`. Most entries in `/proc` and its subdirectories can be read by everyone. The `/proc/net/tcp` file contains information about all TCP connections on an Android device and is also world-readable and hence accessible by every app without requiring any permissions. Information such as source IP and port, destination IP and port and the UID of the process that created the network connection are listed there. Since Android creates a static mapping of Apps to a UID at install time, one

[4] cf. `http://developer.android.com/reference/android/accounts/AccountManager.html` – last access 13.04.2016

can easily learn which app connects to which Internet hosts based on the UID entry in /proc/net/tcp. Having the destination IP for an app's network connection at hand allows an attacker to easily infer to which online service a credential pair is connected by logging all network connections of an app, immediately after a copy operation from a PM to another app was discovered.

Exfiltrating the Data

As we will report in Chapter 6, Page 93, we found that 92.8% of 13,500 popular Android apps request Internet access. Adding the Internet permission to PWSniff should thus not raise undue concern. With this permission, transmitting the harvested data is of course trivial. However, if a zero permission attack app is desired, exfiltration of the harvested data can still be done using another flaw in Android's permission system. Egners et al. [64] describe a loophole in Android's permission system that we adopt for our purposes and which allows PWSniff to send gathered credential information to a remote HTTP server without requiring the Internet permission. After the account login information was gathered, the harvested data is cached until the device's display is turned off. When this happens, an HTTP URL with the following structure is built: `http://<pwsniff-master>/pw#username#service`. This URL is used to invisibly open Android's stock browser when the display is turned off by using Android's PowerManager API. We explicitly call Android's stock browser since some third-party browsers do not hand back control for unknown protocols to the Android OS, which is required to keep the attack stealthy.

The server behind the URL replies with a location header containing a custom protocol, for example: 'Location:pwsniff://all.ok'. Since PWSniff includes an activity that previously registered for the custom pwsniff:// protocol, the browser passes handling for the URI pwsniff://all.ok to PWSniff. Staying invisible, the activity then simply terminates.

After demonstrating how credentials can be sniffed when Android password managers are used, how they can be mapped to online accounts and how this information can be exfiltrated stealthily, the next section gives some relevant excerpts of our detailed security analysis of PMs on Android.

5.4 Security Analysis

We analyzed 13 free and 8 paid Android PM apps in detail (cf. Table 5.4). Our intention was to analyze which apps include the clipboard feature for credential copy & paste, which encryption algorithms protect the password database, whether or not the app includes an embedded browser, whether or not the SD card is used to store the password database and whether or not the app removes itself from the recent apps view. For analysis, we installed all apps on a Samsung Galaxy

Nexus with Android 4.0. We applied forensic techniques[5] to learn database and configuration files' structures of the installed password manager apps. To learn internals of the password managers, we decompiled them[6] and conducted manual static code analysis.

We also conducted static code analysis on the same dataset as in Chapter 6, Page 93 and found that only two apps in this dataset registered for the clipboard change listener. We analyzed both apps manually and found no malicious behaviour in the apps. 907 apps (6.7%) in the sample access the clipboard API programatically to share more complex objects than simple text strings such as images, video or audiofiles.

Free							
App	**Installs**[1]	**EM**	**KD**	**C&P**	**EB**	**SD**	**RA**
PassDroid	100-500k	AES	SHA-256	✓	–	Backup	✓
1Password	100-500k	AES	PBKDF2	✓	✓	Always	✓
KeePassDroid	500k-1m	AES[1]	SHA-256	✓	–	Always	✓
UPM	100-500k	AES	PBE	✓	–	Backup	✓
Pocket	100-500k	AES	PBE	✓	–	Backup	✓
NS Wallet	10-50k	AES	–	✓	–	✓	✓
LastPass	100-500k	AES	○[4]	✓	✓	–	–
PasswdSafe	10-50k	AES	–	✓	–	✓	✓
OI Safe	100-500k	AES	PBE	✓	–	✓	✓
aWallet	100-500k	AES[2]	SHA-256	✓	–	Backup	✓
Moxier Wallet	10-50k	AES	SHA-256	✓	–	–	✓
Keeper	1-5m	AES	SHA-1	✓	✓	Backup	✓
RoboForm	100-500k	○[4]	○[4]	✓	✓	Backup	✓
Paid							
mSecure	100-500k	Blowfish	SHA-256	✓	–	✓	–
Secret Safe	50-100k	AES[3]	SHA-256[3]	✓	–	Backup	✓
SafeWallet	10-50k	AES	HmacSHA1	✓	–	✓	✓
SPB Wallet	10-50k	AES	○[4]	✓	–	✓	✓
eWallet	10-50k	AES	PBE	✓	–	✓	–
Handy Safe Pro	10-50k	Blowfish	–	✓	–	–	✓
DataVault	1-5k	AES	–	✓	–	Backup	✓
Password Box	5-10k	AES	–	✓	–	✓	✓

Table 5.1: Overview of the analysed Android password manager apps. (EM=Encryption Method, KD=Key Derivation, C&P=copy&paste functionality, EB=Embedded Browser, SD=Writes database to SD card, RA=Removes itself from the recent apps view)

[1] KeePassDroid combines AES and Twofish
[2] aWallet combines AES, Blowfish and 3DES
[3] Secret Safe combines AES and Twofish for encryption and multiple rounds of SHA-256 and Whirlpool for key derivation.
[4] This information could not be found by reverse engineering.

[5]We used the adb tool (cf. `http://developer.android.com/tools/help/adb.html` – last access 13.04.2016) for logical extraction.

[6]We used a bundle of decompilation tools: JD-GUI (cf. `http://java.decompiler.free.fr/?q=jdgui` – last access 13.04.2016), apktool (cf. `http://code.google.com/p/android-apktool/` – last access 13.04.2016) and dex2jar (cf. `http://code.google.com/p/android-apktool/` – last access 13.04.2016)

Encryption

One important aspect of PM security is the encryption mechanism to store credential databases. Android's stock browser does not encrypt stored passwords in any way but protects them from unauthorised access by file system permissions. Android's AccountManager mechanism provides centralised credential storage and also protects user credentials from unauthorised access by file system permissions, but the `accounts.db` database is not protected with an extra layer of encryption. This does not protect the password from forensic analysis.

All third party PMs we analyzed apply some encryption mechanism to protect the data. Android supports (3)DES, RC2, and RC5[7] to encrypt data out of the box. Other encryption algorithms require the developer to add third-party libraries to their app. We decompiled the PMs to find out what kind of encryption algorithm is applied in each PM app. To provide stronger security, most password managing apps use the Advanced Encryption Standard (AES) with several key lengths. aWallet uses a combination of AES, Blowfish and 3DES.

A critical aspect of encrypting password databases is the derivation of the encryption key [119] that is directly connected to the master secret used to unlock/decrypt the password database. Seven apps use a dedicated key derivation function to derive the symmetric encryption key from the user's master secret to strengthen the security of encrypted credentials. We found one app that directly inputs the user's password as the encryption key, truncating passwords longer than 16 characters. In case the password has less than 16 characters, the string "FEDCBA9876543210" is appended to *"strengthen"* the password. Another app uses an HMAC algorithm with SHA-256 and the fixed initialization vector `"notverysecretiv"` for key derivation.

Database Structure

Password managers from Table 5.4 differ in the way they store entries in their databases. Some apps such as KeePassDroid store password databases in a single, completely encrypted file[8] and encrypt all data including username, password and online service identification (e. g. the online account's website). 1Password and NS Wallet, on the other side, do not encrypt all information. 1Password puts each entry into a separate JSON-structured file that stores the account's location (i. e. website URL) in cleartext. Passwords and usernames are stored as unsalted hash values. Hence, accounts with equal usernames and/or passwords have the same hash values and enable an attacker to carefully select which hashes might be worth breaking. In their FAQ, the 1Password developers explain that they only encrypt parts of their databases for performance reasons[9].

[7] cf. `http://developer.android.com/re-ference/javax/crypto/-spec/package-summary.html` – last access 13.04.2016

[8] KeePassDroid uses the .kdbx format.

[9] `http://help.agilebits.com/1Password3/agile_keychain_design.html` – last access 13.04.2016

Storage

Most password managers, including Android's stock browser, store password databases in files or SQLite databases that are only accessible by the password manager app itself. Hence, other Android apps cannot access account information regardless of whether it is encrypted or not. Ten of the analysed PMs store databases on the SD card that is world readable without requiring further permissions on all devices with Android 4.0 and older. In combination with inappropriate database structures (not encrypting all information stored in the password manager), an attacker is for instance able to learn for which services a user holds accounts or for which services the same username and/or passwords are used.

Recent Apps

An essential feature of Android devices is an overview of the currently running apps, also called the *Recent Apps View*. The Recent Apps View shows thumbnails of current foreground activities of all running apps. While a security feature of all analysed password managers is the automatic locking of the password database either immediately after the password manager app was left or after a configurable amount of time, we found only three apps that also replace their thumbnails in the recent apps view (cf. Table 5.4). In case the user copied online account information (usually the location, username and password) and then leaves the password manager app to paste the information into another app, the account information is left in the recent apps view and can be seen by anyone with physical access to the user's device. Although this threat is orthogonal to our attack (cf. Section 5.3), it outlines a security risk for users' online credentials.

Cloud Sync

While all password managers store their databases locally and allow synchronisation of their databases, most offer a more manual functionality using Dropbox or similar services. LastPass, SecureSafe and RoboForm provide dedicated Cloud storage features to automatically synchronise all passwords remotely. In Chapter 6, Page 93 we analyzed popular Android apps and found that many app developers fail to apply TLS appropriately, being vulnerable to active Man-In-The-Middle attacks. Although LastPass, SecureSafe and RoboForm protect their network communication with TLS, SecureSafe and RoboForm fail to verify the cloud servers' TLS certificates. Instead, they accept all certificates. In case of SecureSafe this however has not further security implications since in addition to TLS, SecureSafe uses a session-specific symmetric key, which is set up during the SRP-login [203], to additionally encrypt password-data end-to-end. However, RoboForm leaks the users' credentials which are used for password encryption in the default case (i. e. the user did not choose an extra password for encryption). Hence, an attacker can gain access to the data in cleartext under this circumstances.

5.5 Developer Study

After analysing Android password managers on a technical basis, we contacted their developers via email and informed them about a possible security threat for their users. We offered them to get in contact either via email or telephone to discuss the details of the PWSniff attack. We also posed the following questions:

- Why was the C&P feature used in the password manager app?
- Were developers aware of the security threats arising from using the clipboard for username/password sharing, and, if so, why did they add the C&P feature nonetheless?
- Which features, if any, do developers miss in Android's SDK for developing a password manager app?

15 of the 21 developers agreed to participate in the email interview and are anonymously referred to as P1, ..., P15 in the following.

Results

During the discussions with developers, we were able to identify three different reasons to add the usability-enhancing clipboard feature to PM apps. One was because the developers themselves were users of their apps and desired the feature themselves. ("As I'm a [...] program user too, I added the copy feature because I needed to transfer usernames (that are usually long email addresses) and passwords to login forms in web browsers."; P7). The second reason provided by PM developers was the wish to come as close as possible to PM functionality on the desktop, because developers believed that users would reject their apps if they were not sufficiently usable. ("Copy to clipboard has been in [...] Android from early on. [...] It was something that we knew we needed to make the application usable at all"; P4). Lastly, developers reported that users directly requested a C&P feature for their app ("The feature was highly requested by users. The most common example: users want to login to a website on their mobile device, so he/she copies credentials from [our PM] to the clipboard and then pastes them into the browser."; P15).

All but one developer were aware of security threats resulting from putting passwords into a device's clipboard. Developers who were aware of the security threat justified adding the clipboard integration, stating that they had no other choice. They described it as a tradeoff between usability and security which was decided in favour of increasing usability ("It's a balance between ease of use and security. Of course it would be much more secure to not use the clipboard, however people accept the risk of doing so; the alternative of not using a password manager is worse."; P3). One developer interestingly described his decision not as a usability-security tradeoff but as a "one type of security versus another type of security" decision, alluding to the fact that without password managers users would choose less secure passwords. Additionally, P4 stated: "On the whole, I think that password reuse [...]

is currently the biggest single problem with password security today. And so, if a password manager gets people to use unique passwords for each site, the dangers of a publicly readable clipboard is a security risk that can be worthwhile. [...] What's the alternative?".

All developers criticised Android's missing support for password manager apps. A native integration into third party apps and browsers was described as the most effective countermeasure against the password sniffing security threat ("Android doesn't offer hooks into the native default browser [...] and does not allow our app to access input fields of other apps [...] which makes it necessary that password managers make heavy use of the clipboard."; P3).

Based on the lack of API support for third-party password managers on the Android OS, developers decided to opt for the best usability they could achieve by including the clipboard feature to allow users to copy-and-paste usernames and passwords from their apps to other apps. Although all but one developer were aware of the possible security threat, they decided that better usability was more important than stronger security. A justification multiple developers offered was that *they had no other choice* and that it was necessary to add the best possible usability even if security was threatened.

5.6 Countermeasures

With the results of our analysis and the developers' comments in mind, we first discuss possible countermeasures to improve the security of a smartphone's clipboard facilities as a global shared memory. Additionally, we present a PM implementation for Android based on a customised soft-keyboard that provides usability features similar to desktop PMs and does not leak credentials over public channels.

Secure Clipboard Architecture

Sniffing confidential information on Android devices is currently easy since on the one hand, a proper plugin API for integrating password managers is missing and, on the other hand, the design of the current clipboard mechanism on Android is not made for sharing confidential information between apps. The current clipboard model allows an arbitrary app to access clipboard items deposited by any other app. With the assumption that both, the copy as well as the paste operation are triggered by the user, such a clipboard model does not cause security concerns. However, on Android, two other API features open the door for malicious activity: Android's background service feature for apps and the `ClipboardManager.OnPrimaryClipChangedListener` allow for stealthy harvesting of clipboard items (cf. Section 5.3). Therefore, we present two possible modifications to improve Android's clipboard model when it is accessed using API functionalities:

Permissions The current clipboard model allows every app to programmatically read data from and write data to the clipboard, without requiring permission for that. While user-triggered clipboard operations can remain unchanged,

we propose two new permissions for API-based access to clipboard functionality: `WRITE_TO_CLIPBOARD` and `READ_FROM_CLIPBOARD`. Although the limited effects of Android's permission model for the average app user have been discussed (cf. Felt et al. [84]), these permissions should be added for completeness. This way, at least the tech-savvy users would have a chance to see if an app is capable of accessing the clipboard programmatically and can warn the rest of the community. Since we identified only very few apps to access the clipboard programatically (cf. Section 5.4), the proposed changes would only impact a small number of apps. Regular, user-triggered copy-and-paste operations would not be influenced by this modification.

Targeted Clipboard Copying a value to the clipboard on current Android smartphones is equivalent to broadcasting the information to all other apps. This is contrary to the users' intuition of using a copy-and-paste feature that is generally used to transfer information from one app to another. Therefore we propose to extend API calls to the clipboard with a "target app" parameter that the app may request from the user. Keeping usability in mind, the number of target apps should be kept to a minimum. Apps providing an API-based copy feature may let the user choose target apps from a list of all apps or suggest useful targets as well as remember previous preferences. If clipboard operations are triggered by the user, reading the clipboard's contents should only be possible through explicit user interaction as well.

The modifications to Android's current clipboard model proposed above do not only protect credentials from unwanted disclosure, but can also serve to shield any other (possibly confidential) information (such as financial or medical information), that a user might copy to the clipboard.

USecPassBoard

While the above solutions would alleviate the current security problems of PMs, they would also require modification of the Android OS itself. Additionally, these measures cannot address the usability issues of mobile PMs, i. e. that the user needs to manually select credentials, switch apps and manually paste. To offer both better security *and* usability we propose a novel password manager: USecPassBoard. To overcome the issues plaguing the traditional approach of mobile password managers, we went down a different path. We created a soft-keyboard which integrates a password manager. Since soft-keyboards are available in every app and can access a shared credential database, they integrate well with most scenarios where credentials need to be entered. A custom soft-keyboard implementation on Android replaces the default keyboard and provides a custom means to input data into user-input fields. Figure 5.1 shows the user interface of the USecPassBoard PM. Besides preserving the regular keyboard functionality, it essentially adds two operations: (1) Creating a new username/password entry and (2) inserting a username/password tuple at the user's discretion. Since USecPassBoard is a soft-keyboard, it is available in every application, including the browser and stores passwords in a master secret-

protected AES-256 encrypted database[10] to protect username/password tuples from unauthorized access. This effectively avoids the use of copy-and-paste on usernames and passwords while maintaining the flexibility of all available password managers.

5.7 USecPassBoard User Interface

New Account

USecPassBoard analyses the context of user input to determine if credentials are being entered. The password context is determined by identifying which app is currently used (i. e. which app is in the foreground) and in case the foreground app is a browser, it determines the website which is displayed by reading the browser's first item cached in the history. Apps are uniquely identified based on their package names managed by the Android operating system[11]. USecPassBoard then caches the input of all textfields in the foreground activity. This is possible since soft-keyboards on Android are triggered when a textfield is activated by the user. Additionally, a soft-keyboard receives a reference of the `EditorInfo` class[12] which identifies an input as a text or a password field. After the user completes the input and the keyboard loses the focus on a password field, a notification is displayed in the status bar that a new dataset was created (cf. Figure 5.1) if there is no identical username/-password tuple for the current context in the database. New username/password tuples are bound to the target app – based on its package name – and are not available for possibly malicious apps. In case the user would like to share credentials between different apps (e. g. between two Facebook client apps), we allow this in the settings menu of USecPassBoard.

Credential Insertion

In case USecPassBoard recognises a known password context (i. e. a package name of an app for which credentials are stored in the database), the user can choose to insert this information by tapping into the input field for the username or password. A popup message appears after the user tapped onto the key button (cf. Figure 5.1) and a list of available credentials for the given password context is displayed. Subsequently the selected username/password tuple is inserted at the user's discretion and the login process can be started.

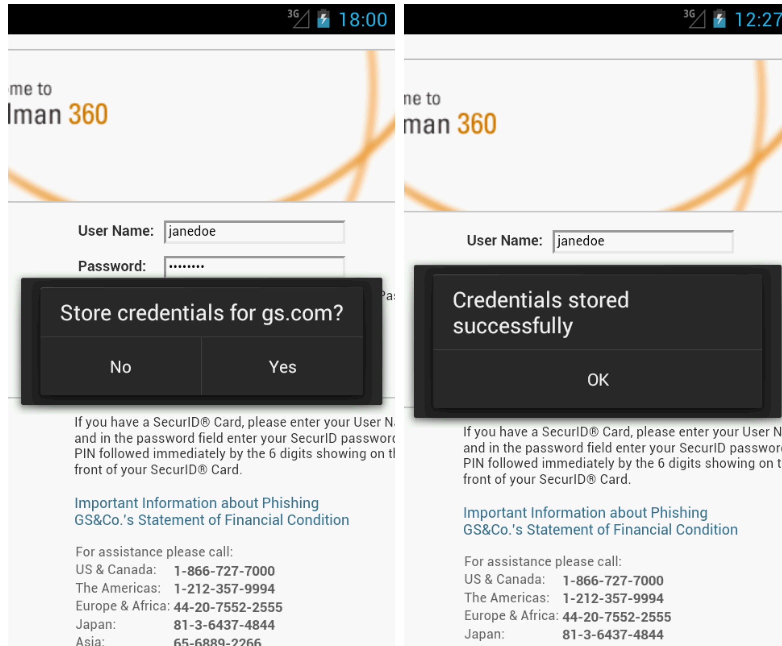
Security Considerations

All interactions between the USecPassBoard virtual keyboard and a target app must be initiated by the user by tapping into a text input field. This creates a communi-

[10] We use the SQLCipher (cf. `http://sqlcipher.net/sqlcipher-for-android/` – last access 13.04.2016) database.

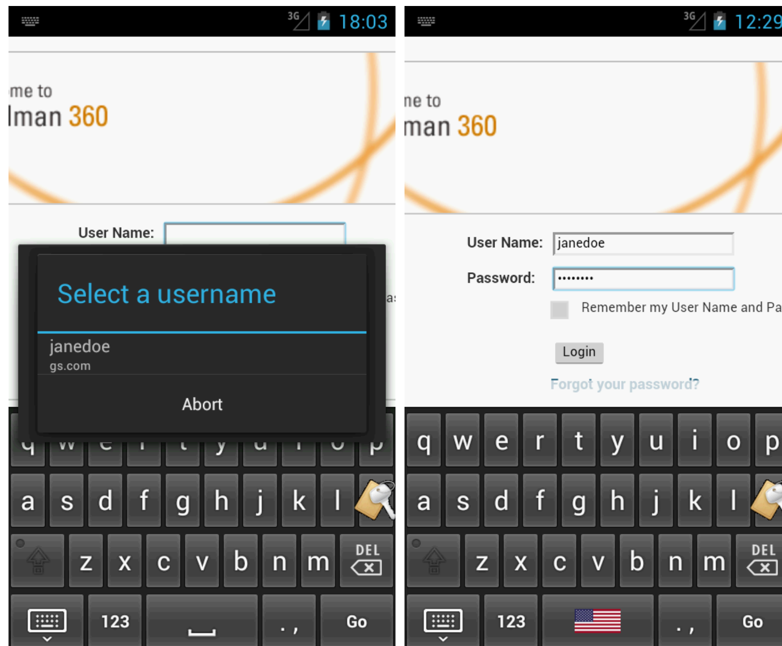
[11] cf. `http://developer.android.com/guide/topics/manifest/manifest-element.html` – last access 13.04.2016

[12] cf. `https://developer.android.com/reference/android/view/inputmethod/EditorInfo.html` – last access 13.04.2016



(a) Asking the user to store new credential tuple.

(b) Successfully stored new credential tuple.



(c) Selecting existing credential tuple.

(d) Credentials filled in.

Figure 5.1: The USecPassBoard workflow for storing new credential tuples and filling in stored credentials.

cation channel between the keyboard and the target app through Android's Input-MethodManager[13] which is not accessible from other third party apps. This allows the automatic storage of new account credentials and insertion of stored credentials into uniquely identifiable target apps.

Target apps are uniquely identified based on their package name that is managed by the Android OS and cannot be spoofed by malicious apps[14]. In case the target app is the browser, a password context consists of the browser's package name and a target website. We identify the target website by reading the top item from the browser's history. This is accessible with Android's READ_HISTORY_BOOKMARKS permission and gives us the currently viewed website. Hence, we can avoid that users falsely insert credentials another website.

Since the focus of this work was on investigating developers' challenges when working with security relevant APIs, we did not evaluate the security and usability of the USecPassBoard solution. This was out of scope of this work. However, after we notified password manager developers of the copy & paste security issue, the developers of LastPass (cf. Table 5.4) implemented a custom keyboard similar to our solution.

5.8 Summary

With the rise of mobile devices, mobile password manager apps could be an integral security tool for smartphone and tablet PC users. Since Android based devices lack APIs for the integration of password managers, current solutions rely heavily on the clipboard to share credentials between the PM and other apps. We analysed 21 popular password managers on Android which all are vulnerable to credential sniffing because a device's clipboard is a publicly available storage that can be accessed from any app. We showed that, using additional context information, malware is able to link the stolen credentials to the corresponding online account in many cases. We interviewed developers of the analysed PM apps and found that the majority of them were aware of possible security threats but accepted the risk to provide better usability. Based on the analyses' findings and developers' feedback, we discuss modifications to Android's clipboard mechanism to increase security for sensitive information. Finally, we present a soft-keyboard that integrates a secure and easy to use password manager which prevents the leakage of usernames/passwords via the clipboard. This password manager design is the first to offer both usability and security for Android-based password managers.

This chapter investigated how Android app developers work around security re-

[13] cf. `http://developer.android.com/reference/android/view/inputmethod/InputMethodManager.html` – last access 13.04.2016

[14] cf. `http://source.android.com/tech/security/` – last access 13.04.2016

lated APIs that provide an unsafe default, i. e. using the clipboard to copy passwords from a password manager into another app leaks passwords. Working around this would require developers to implement a more secure solution than available by default. However, even after we informed developers of the vulnerability, only a small fraction started working on a more secure solution that circumvented the system's insecure API. While a security-focused, experienced developer should be able to find a usable solution for this class of API issue, the insecure default sets the bar high for the average developer.

Hence, looking at the use of security related APIs for safe defaults could provide interesting insights into how their usability affects the security of software. To investigate this circumstance, the next chapter describes analyses of how Android app developers work with X.509 certificates to secure network connections. While the Android APIs provides a safe default, developers have the freedom to work around these defaults to implement exotic use cases. The intention of this flexible API was to be able to implement more restrictive certificate validation such as public key pinning. However, developers can misuse the API to turn off certificate validation entirely and weaken the security of their apps.

The next chapter offers a more complete picture of how developers work with security APIs, both APIs providing safe defaults and APIs not providing safe defaults.

6 Developers: Customizing Certificate Validation

Disclaimer: *The contents of this chapter were previously published as part of the paper "Why Eve and Mallory Love Android: An Analysis of Android SSL (In)Security" presented at* 19th *ACM Conference on Computer and Communications Security (CCS) in 2012 [76] together with my co-authors Lars Baumgärtner, Bernd Freisleben, Marian Harbach, Thomas Muders and Matthew Smith. As this publication was joint work with my co-authors as a team, this chapter will use the academic "we" to mirror this fact. The idea and initial concept for this work came from myself. I developed the static analysis tool "MalloDroid" that was I to analyze 13,500 popular Android apps for their correct usage of TLS certificate validation. I personally analyzed most of the apps we manually checked. The user study to measure the effectiveness of Android's browser TLS warning message was designed and conducted by myself. However, my co-authors contributed in different ways. Thomas Muders assisted in manually analyzing potentially vulnerable Android apps. Lars Baumgärtner reverse engineered the Zoner Anti-Virus app. Marian Harbach assisted in analyzing the user study results and together with Bernd Freisleben and Matthew Smith helped in compiling the paper for publication.*
Software described in this chapter is available at `https://zenodo.org/record/50526`.

6.1 Motivation

Unlike the "walled garden" approach of Apple's App Store, Android software development and the Google Play Market are relatively open and unrestricted. This offers both developers and users more flexibility and freedom, but also creates significant security challenges. The coarse permission system [69] and over-privileging of applications [159] can lead to exploitable applications. Consequently, several efforts have been made to investigate privilege problems in Android apps [84, 69, 49, 31, 162]. Enck et al. introduced TaintDroid [66] to track privacy-related information flows to discover such (semi-)malicious apps. Bugiel et al. [31] showed that colluding malicious apps can facilitate information leakage. Furthermore, Enck et al. analyzed 1,100 Android apps for malicious activity and detected widespread use of privacy-related information such as IMEI, IMSI, and ICC-ID for "cookie-esque" tracking. However, no other malicious activities were found, in particular no exploitable vulnerabilities that could have lead to malicious control of a smartphone were observed [67].

In this work, instead of focusing on malicious apps, we investigate potential security threats posed by benign Android apps that legitimately process privacy-related user data, such as log-in credentials, personal documents, contacts, financial data, messages, pictures or videos. Many of these apps communicate over the Internet for legitimate reasons and thus request and require the INTERNET permission. It is then necessary to trust that the app adequately protects sensitive data when transmitting via the Internet.

The most common approach to protect data during communication on the Android platform is to use the TLS protocols. This work is a continuation of the work in

Chapter 4, page 59 which focused on system administrators of HTTPS enabled webservers. In contrast to system administrators, developers of Android apps have a different view on the TLS ecosystem. Apps written by developers interact with webservers operated by system administrators and have to handle (possibly insecure) configurations. This circumstance allows us to investigate a very important question: Are developers able to deal with real world TLS deployments provided by system administrators?

To evaluate this question in the context of the state of TLS use in Android apps, we downloaded 13,500 popular free apps from Google's Play Market and studied their properties with respect to the usage of TLS. In particular, we analyzed the apps' vulnerabilities against MITM attacks due to the inadequate or incorrect use of TLS.

For this purpose, we created MalloDroid, an Androguard[1] extension that performs static code analysis to a) analyze the networking API calls and extract valid HTTP(S) URLs from the decompiled apps; b) check the validity of the TLS certificates of all extracted HTTPS hosts; and c) identify apps that contain API calls that differ from Android's default TLS usage, e. g., contain non-default trust managers, `SSLSocketFactory` or `HostnameVerifier` implementations with permissive verification strategies. Based on the results of the static code analysis, we selected 100 apps for manual audit to investigate various forms of TLS use and misuse: accepting all TLS certificates, allowing all hostnames regardless of the certificate's Common Name (CN), neglecting precautions against TLS stripping, trusting all available Certificate Authorities (CAs), not using public key pinning, and misinforming users about TLS usage.

Furthermore, we studied the visibility and awareness of TLS security in the context of Android apps. In Android, the user of an app has no guarantee that an app uses TLS and also gets no feedback from the Android operating system whether TLS is used during communication or not. It is entirely up to the app to use TLS and to (mis)inform the user about the security of a network connection. However, even when apps present warnings and security indicators, users need to see and interpret them correctly. The users' perceptions concerning these warnings and indicators were investigated in an online survey. Finally, several countermeasures that could help to alleviate the problems discovered in the course of our work are discussed.

The results of our investigations can be summarized as follows:

- 1,074 apps contain TLS specific code that either accepts all certificates or all hostnames for a certificate and thus are potentially vulnerable to MITM attacks.
- 41 of the 100 apps selected for manual audit were vulnerable to MITM attacks due to various forms of TLS misuse.
- The cumulative install base of the apps with confirmed vulnerabilities against MITM attacks lies between 39.5 and 185 million users, according to Google's

[1] cf. `https://github.com/androguard/androguard` – last access 13.04.2016

Play Market.[2] This number includes 3 apps with install bases between 10 and 50 million users each.

- From these 41 apps, we were able to capture credentials for American Express, Diners Club, Paypal, bank accounts, Facebook, Twitter, Google, Yahoo, Microsoft Live ID, Box, WordPress, remote control servers, arbitrary email accounts, and IBM Sametime, among others.
- We were able to inject virus signatures into an anti-virus app to detect arbitrary apps as a virus or disable virus detection completely.
- It was possible to remotely inject and execute code in an app created by a vulnerable app-building framework.
- 378 (50.1%) of the 754 Android users participating in the online survey did not judge the security state of a browser session correctly.
- 419 (55.6%) of the 754 participants had not seen a certificate warning before and typically rated the risk they were warned against as medium to low.

6.2 Background

Android & TLS

The Android SDK offers several convenient ways to access the network. The `java.net`, `javax.net`, `android.net` and `org.apache.http` packages can be used to create (server) sockets or HTTP(S) connections. The `org.webkit` package provides access to web browser functionality. In general, Android allows apps to customize TLS usage – i. e., developers must ensure that they use TLS correctly for the intended usage and threat environment. Hence, the following (mis-)use cases can arise and can cause an app to transmit sensitive information over a potentially broken TLS channel:

Trusting all Certificates. The TrustManager interface can be implemented to trust all certificates, irrespective of who signed them or even for what subject they were issued.

Allowing all Hostnames. It is possible to forgo checks of whether the certificate was issued for this address or not, i. e., when accessing the server `example.com`, a certificate issued for `some-other-domain.com` is accepted.

Trusting many CAs. This is not necessarily a flaw, but Android 4.0 trusts 134 CA root certificates per default. Due to the attacks on several CAs in 2011, the problem of the large number of trusted CAs is actively debated.[3]

[2] Google's Play Market only gives a range within which the number of installed apps lies based on the installs from the Play Market. The actual number is likely to be larger, since alternative app markets for Android also contribute to the install base.

[3] cf. `http://android-ssl.org/s/5` – last access 13.04.2016

Mixed-Mode/No TLS. App developers are free to mix secure and insecure connections in the same app or not use TLS at all. This is not directly an TLS issue, but it is relevant to mention that there are no outward signs and no possibility for a common app user to check whether a secure connection is being used. This opens the door for attacks such as TLS stripping (cf. Chapter 2.3.3, Page 27) or tools like Firesheep.[4]

On the other hand, Android's flexibility in terms of TLS handling allows advanced features to be implemented. One important example is TLS Pinning[5], in which either a (smaller) custom list of trusted CAs or even a custom list of specific certificates is used. Android does not offer TLS pinning capabilities out of the box. However, it is possible to create a custom trust manager to implement TLS pinning.[6]

Android Security

There have been several efforts to investigate Android permissions and unwanted or malicious information flows, such as the work presented by Enck et al. [69, 66], Felt et al. [84, 159], Davi et al. [49], Bugiel et al. [31], Nauman et al. [153] and Egners et al. [63].

There are several good overviews of the Android security model and threat landscape, such as Vidas et al. [196], Shabatai [175] et al. and Enck et al. [68, 67]. These papers do not discuss the vulnerability of TLS or HTTPS on Android. Enck et al. [67] does mention that some apps use sockets directly, bearing the potential for vulnerabilities, but no malicious use was found (cf. [67], Finding 13). Our investigation shows that there are several TLS-related vulnerabilities in Android apps, endangering millions of users.

McDaniel et al. [151] and Zhou et al. [209] also mainly focus on malicious apps in their work on the security issues associated with the app market model of software deployment. The heuristics of DroidRanger [209] could be extended to detect the vulnerabilities uncovered in our work.

While previous works in the field of Android security investigated different areas, our work is the first to analyze the security of custom TLS code in third party Android apps.

TLS Security

A good overview of current TLS problems can be found in Marlinspike's Black Hat talks [142, 143]. The talks cover issues of security indicators, Common Name (CN) mismatches and the large number of trusted CAs and intermediate CAs. Marlinspike

[4]Firesheep is a browser extension that uses a packet sniffer to intercept unencrypted cookies from websites. The tool was released to demonstrate the security risk of session hijacking vulnerabilities to users of web sites that only use HTTPS for the login and do not encrypt the cookie(s) created during the login process. (cf. `https://codebutler.com/firesheep` – last access 13.04.2016)

[5]cf. `http://android-ssl.org/s/6` – last access 13.04.2016

[6]cf. `http://android-ssl.org/s/7` – last access 13.04.2016

also introduces the TLS stripping attack. The fact that many HTTPS connections are initiated by clicking a link or via redirects is particularly relevant for mobile devices, since the MITM attack needed for TLS stripping is easier to execute [182, 116] and the visual indicators are hard to see on mobile devices. More information on TLS can be found in Chapter 2.3.3 on Page 27.

Shin et al. [180] study the problem of TLS stripping for desktop browsers and present a visual-security-cue-based approach to hinder TLS stripping in this environment. They also highlight the particular problem of this type of attack in the mobile environment and suggest that it should be studied in more detail.

Egelman et al. [60] and Sunshine et al. [188] both study the effectiveness of browser warnings, showing that their effectiveness is limited and that there are significant usability issues. Although both of these studies were conducted in a desktop environment, the same caveats need to be considered for mobile devices. In this work, we conduct a first online survey to gauge the awareness and effectiveness of browser certificate warnings and HTTPS visual security indicators on Android.

6.3 Evaluating Android TLS Usage

Our study of Android TLS security encompasses popular free apps from Google's Play Market. Overall, we investigated 13,500 applications. We built MalloDroid, an extension of the Androguard reverse engineering framework, to automatically perform the following steps of static code analysis:

Permissions. MalloDroid checks which apps *request the INTERNET permission*, which apps *actually contain INTERNET permission-related API calls* and which apps *additionally request and use privacy-related permissions* (cf. [159]).

Networking API Calls. MalloDroid analyzes the use of *HTTP transport* and *Non-HTTP transport* (e. g., direct socket connections).

HTTP vs HTTPS. MalloDroid checks the validity of URLs found in apps and groups the apps into *HTTP only*, *mixed-mode (HTTP and HTTPS)* and *HTTPS only*.

HTTPS Available. MalloDroid tries to establish a secure connection to HTTP URLs found in apps.

Deployed Certificates. MalloDroid downloads and evaluates TLS certificates of hosts referenced in apps.

TLS Validation. MalloDroid examines apps with respect to inadequate TLS validation (e. g., apps containing code that allows all hostnames or accepts all certificates).

12,534 (92.84%) of the apps in our test set request the network permission `android.permission.INTERNET`. 11,938 (88.42%) apps actually perform networking related API calls. 6,907 (51.16%) of the apps in our sample use the INTERNET

permission in addition to permissions to access privacy related information such as the users' calendars, contacts, browser histories, profile information, social streams, short messages or exact geographic locations. This subset of apps has the potential to transfer privacy-related information via the Internet. This subset does not include apps such as banking, business, email, social networking or instant messaging apps that intrinsically contain privacy-relevant information without requiring additional permissions.

We found that 91.7% of all networking API calls are related to HTTP(S). Therefore, we decided to focus our further analysis on the usage of HTTP(S). To find out whether an app communicates via HTTP, HTTPS, or both, MalloDroid analyzes HTTP(S) specific API calls and extracts URLs from the decompiled apps.

HTTP vs. HTTPS

MalloDroid extracted 254,022 URLs. It can be configured to remove certain types of URLs for specific analysis. For this study, we removed 58,617 URLs pointing to namespace descriptors and images, since these typically are not used to transmit sensitive user information. The remaining 195,405 URLs pointed to 25,975 unique hosts. 29,685 of the URLs (15.2%) pointing to 1,725 unique hosts (6.6%) are HTTPS URLs. We further analyzed how many of the hosts referenced in HTTP URLs could also have been accessed using HTTPS.

76,435 URLs (39.1%) pointing to 4,526 hosts (17.4%) allowed a valid HTTPS connection to be established, using Android's default trust roots and validation behavior of current browsers. This means that 9,934 (73.6%) of all 13,500 tested apps could have used HTTPS instead of HTTP with minimal effort by adding a single character to the target URLs. We found that 6,214 (46%) of the apps contain HTTPS and HTTP URLs simultaneously and 5,810 (43%) do not contain HTTPS URLs at all. Only 111 apps (0.8%) exclusively contained HTTPS URLs.

For a more detailed investigation, we looked at the top 50 hosts, ranked by the number of occurrences. This group mainly consists of advertising companies and social networking sites. These two categories account for 37.9% of the total URLs found, and the hosts are contained in 9,815 (78.3%) of the apps that request the INTERNET permission.

Table 6.3 presents an overview of the top 10 hosts. The URLs pointing to the these hosts suggest they are often used for Web Service API calls, authentication and fetching/sending user or app information. Especially in the case of ad networks that collect phone identifiers and geolocations [66] and social networks that transport user-generated content, the contained information is potentially sensitive.

34 of the top 50 hosts offer all their API calls via HTTPS, but none is accessed exclusively via HTTPS. Of all the URLs pointing to the top 50 hosts, 22.1% used HTTPS, 61% could have used HTTPS by substituting *http://* with *https://*, and 16.9% had to use HTTP because HTTPS was not available. The hosts `facebook.com` and `tapjoyads.com` are positive examples, since the majority of the URLs we found for these two hosts already use HTTPS.

Host	has TLS	# URLs	# HTTPS
market.android.com	✓	6,254	3,217
api.airpush.com	✓	5,551	0
a.admob.com	✓	4,299	0
ws.tapjoyads.com	✓	3,410	3,399
api.twitter.com	✓	3,220	768
data.flurry.com	✓	3,156	1,578
data.mobclix.com	✓	2,975	0
ad.flurry.com	✓	2,550	0
twitter.com	✓	2,410	129
graph.facebook.com	✓	2,141	1,941

Table 6.1: The top 10 hosts used in all extracted URLs and their TLS availability, total number of URLs and number of HTTPS URLs pointing to that host.

Deployed TLS Certificates

To analyze the validity of the certificates used by HTTPS hosts, we downloaded the TLS certificates for all HTTPS hosts extracted from our app test set, yielding 1,887 unique TLS certificates. Of these certificates, 162 (8.59%) failed the verification of Android's default TLS certificate verification strategies, i. e., 668 apps contain HTTPS URLs pointing to hosts with certificates that could not be validated with the default strategies. 42 (2.22%) of these certificates failed TLS verification because they were self-signed, i. e., HTTPS links to self-signed certificates are included in 271 apps. 21 (1.11%) of these certificates were already expired, i. e., 43 apps contain HTTPS links to hosts with expired TLS certificates.

For hostname verification, we applied two different strategies that are also available in Android: the *BrowserCompatHostnameVerifier*[7] and the *StrictHostnameVerifier*[8] strategy. We found 112 (5.94%) certificates that did not pass strict hostname verification, of which 100 certificates also did not pass the browser compatible hostname verification. Mapping these certificates to apps revealed that 332 apps contained HTTPS URLs with hostnames failing the *BrowserCompatHostnameVerifier* strategy.

Overall, 142 authorities signed 1,887 certificates. For 45 (2.38%) certificates, no valid certification paths could be found, i. e., these certificates were signed by authorities not reachable via the default trust anchors. These certificates are used by 46 apps. All in all, 394 apps include HTTPS URLs for hosts that have certificates that are either expired, self-signed, have mismatching CNs or are signed by non-default-trusted CAs.

Custom TLS Validation

Using MalloDroid, we found 1,074 apps (17.28% of all apps that contain HTTPS URLs) that include code that either bypasses effective TLS verification completely by

[7] cf. `http://android-ssl.org/s/8` – last access 13.04.2016
[8] cf. `http://android-ssl.org/s/9` – last access 13.04.2016

accepting all certificates (790 apps) or that contain code that accepts all hostnames for a certificate as long as a trusted CA signed the certificate (284 apps).

While an app developer wishing to accept all TLS certificates must implement the TrustManager interface and/or extend the SSLSocketFactory class, allowing all hostnames only requires the use of the `org.apache.http.conn.ssl.AllowAllHostnameVerifier` that is included in 453 apps. Additionally, MalloDroid found a `FakeHostnameVerifier`, `NaiveHostnameVerifier` and `AcceptAllHostnameVerifier` class that can be used in the same way.

To understand how apps use "customized" TLS implementations, we searched for apps that contain non-default trust managers, TLS socket factories and hostname verifiers differing from the `BrowserCompatHostnameVerifier` strategy. We found 86 custom trust managers and TLS socket factories in 878 apps. More critically, our analysis also discovered 22 classes implementing the `TrustManager` interface and 16 classes extending the `SSLSocketFactory` that accept all TLS certificates. Table 6.3 shows which broken trust managers and TLS socket factories were found.

Trust Managers	**Socket Factories**
AcceptAllTrustM	AcceptAllSSLSocketF
AllTrustM	AllTrustingSSLSocketF
DummyTrustM	AllTrustSSLSocketF
EasyX509TrustM	AllSSLSocketF
FakeTrustM	DummySSLSocketF
FakeX509TrustM	EasySSLSocketF
FullX509TrustM	FakeSSLSocketF
NaiveTrustM	InsecureSSLSocketF
NonValidatingTrustM	NonValidatingSSLSocketF
NullTrustM	NaiveSslSocketF
OpenTrustM	SimpleSSLSocketF
PermissiveX509TrustM	SSLSocketFUntrustedCert
SimpleTrustM	SSLUntrustedSocketF
SimpleX509TrustM	TrustAllSSLSocketF
TrivialTrustM	TrustEveryoneSocketF
TrustAllManager	NaiveTrustManagerF
TrustAllTrustM	LazySSLSocketF
TrustAnyCertTrustM	UnsecureTrustManagerF
UnsafeX509TrustM	
VoidTrustM	

Table 6.2: Trust Managers & Socket Factories that trust all certificates (suffixes omitted to fit the page)

This small number of critical classes affects a large number of apps. Many of the above classes belong to libraries and frameworks that are used by many apps. 313 apps contained calls to the `NaiveTrustManager` class that is provided by a crash report library. In 90 apps, MalloDroid found the `NonValidatingTrustManager` class provided by an SDK for developing mobile apps for different platforms with just a single codebase. The `PermissiveX509TrustManager`, found in a library for sending different kinds of push notifications to Android devices, is included in 76 apps. Finally, in 78 apps, MalloDroid found a `SSLSocketFactory` provided by

a developer library that accepts all certificates. The library is intended to support developers to write well designed software and promotes itself as a library for super-easy and robust networking. Using any of the above Trust Managers or Socket Factories results in the app trusting all certificates.

Manual Investigation

The static code analysis presented above only shows the potential for security problems. The fact that code for insecure TLS is present in an app does not necessarily mean that it is used or that sensitive information is passed along it. Even more detailed automated code analysis, such as control flow analysis, data flow analysis, structural analysis and semantic analysis cannot guarantee that all uses are correctly identified [67]. Thus, we decided to conduct a more detailed manual study to find out what sort of information is actually sent via these potentially broken TLS communication channels, by installing apps on a real phone and executing an active MITM attack against the apps. For this part of the study, we narrowed our search down to apps from the Finance, Business, Communication, Social and Tools categories, where we suspected a higher amount of privacy relevant information and a higher motivation to protect the information. In this test set, there are 266 apps containing broken TLS or hostname verifiers (Finance: 45, Social: 94, Communication: 49, Business: 60, Tools: 18). We ranked these apps based on their number of downloads and selected the top 100 apps for manual auditing. Additionally, we cherry-picked 10 high profile apps (large install base, popular services) that contained no TLS-related API calls but contained potentially sensitive information, to see whether this information was actually sent in the clear or whether some protection mechanism other than TLS was involved. Subsequently, we analyze the actual impact of insecure certificate verification implementations on users of those Android apps.

Test Environment

For the manual app auditing, we used a Samsung Galaxy Nexus smartphone with Android 4.0 Ice Cream Sandwich. We installed the potentially vulnerable apps on the phone and set up a Wi-Fi access point with a MITM TLS proxy. Depending on the vulnerability to be examined, we equipped the TLS proxy either with a self-signed certificate or with one that was signed by a trusted CA, but for an unrelated hostname.

Of the 100 apps selected for manual audit, 41 apps proved to have exploitable vulnerabilities. We could gather bank account information, payment credentials for PayPal, American Express and others. Furthermore, Facebook, email and cloud storage credentials and messages were leaked, access to IP cameras was gained and control channels for apps and remote servers could be subverted. According to Google's Play Market, the combined install base of the vulnerable apps in our test set of 100 apps was between 39.5 and 185 million users at the time of writing. In the following, we briefly discuss our findings to illustrate the scope of the problem.

Trusting All Certificates

21 apps among the 100 selected apps were vulnerable to this attack. We gave our MITM attack proxy a self-signed certificate for the attack. The apps leaked information such as login credentials, webcam access or banking data. One noteworthy contender was a generic online banking app. The app uses separate classes for each bank containing different trust manager implementations. 24 of the 43 banks supported were not protected from our MITM attack. The app also leaks login credentials for American Express, Diners Club and Paypal. The Google Play Market reports an install base between 100,000 and half a million users. A further app vulnerable to this attack offers instant messaging for the Windows Live Messenger service. The app has an install base of 10 to 50 million users and is listed in the top 20 apps for the communication category in the Google Play Market (as of April 30th, 2012). Username and password are both sent via a broken TLS channel and were sniffed during our attack. This effectively gives an attacker full access to a Windows Live account that can be used for email, messaging or Microsoft's SkyDrive cloud storage. We also found a browser with an install base between 500,000 and one million users that trusts all certificates. The browser does not correctly handle TLS at all, i. e., it accepts an arbitrary certificate for every website the user visits and hence leaks whatever data the user enters. All three apps do not provide any TLS control or configuration options for the user. None of the other apps vulnerable to this attack showed warning messages to the user while the MITM attack was being executed.

Allowing All Hostnames

Next, we found a set of 20 apps that accepted certificates irrespective of the subject name, i. e., if the app wants to connect to `https://www.paypal.com`, it would also accept a certificate issued to `some-domain.com`. We used a certificate for an unrelated domain signed by startSSL[9] for our attacks in this category. The apps leaked information such as credentials for different services, emails, text messages, contact data, bitcoin-miner API keys, premium content or access to online meetings. A particularly interesting finding was an anti-virus app that updated its virus signatures file via a broken TLS connection. Since it seems that the connection is considered secure, no further validation of the signature files is executed by the app. Thus, we were able to feed our own signature file to the anti-virus engine. First, we sent an empty signature database that was accepted, effectively turning off the anti-virus protection without informing the user. In a second attack, we created a virus signature for the anti-virus app itself and sent it to the phone. This signature was accepted by the app, which then recognized itself as a virus and recommended to delete itself, which it also did. Figure 6.1 shows a screenshot of the result of this attack. This is a very stark reminder that defense in depth is an important security principle. Since the TLS connection was deemed secure, no further checks were performed to determine whether the signature files were legitimate. The app

[9] cf. `https://www.startssl.com/` – last access 13.04.2016

has an install base of 500,000 to one million users.[10]

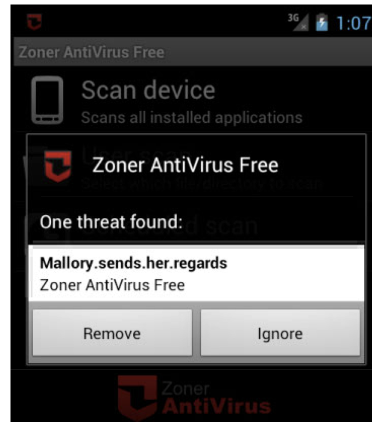


Figure 6.1: After injecting a virus signature database via a MITM attack over broken TLS, the AntiVirus app recognized itself as a virus and recommended to delete the detected malware.

A second example in this category is an app that offers *"Simple and Secure"* cloud-based data sharing. According to the website, the app is used by 82% of the FORTUNE 500 companies to share documents. It has an install base between 1 and 5 million users. While the app offers simple sharing, it leaks the login credentials during the MITM attack. One interesting finding in this app was that the login credentials were leaked from a broken TLS channel while up- and downloads of files were properly secured. However, using the login credentials obtained from the broken channel is sufficient to hijack an account and access the data anyway.

A third example is a client app for a popular Web 2.0 site with an install base of 500,000 to 1 million users. When using a Facebook or Google account for login, the app initiates OAuth login sequences and leaks Facebook or Google login credentials.

We also successfully attacked a very popular cross-platform messaging service. While the app has been criticized for sending messages as plaintext and therefore enabling Eve to eavesdrop, the TLS protection that was intended to secure 'sensitive' information such as registration credentials and the user's contact does not protect from Mallory. For instance, we were able to obtain all telephone numbers from a user's address book using a MITM attack. At the time of writing, the app had an install base of 10 to 50 million users.

TLS Stripping

TLS stripping can occur if a browsing session begins using HTTP and switches to HTTPS via a link or a redirect. This is commonly used to go to a secure login page from an insecure landing page. The technique is mainly an issue for Android browser apps, but it can also affect other apps using Android's `webkit.WebView` that do

[10]honored as the *"Best free anti-virus program for Android" with a detection rate* > 90% *–* `http://www.av-test.org/en/tests/android/` *– last access 13.04.2016*

not start a browsing session with a HTTPS site. We found the `webkit.WebView` in 11,038 apps. Two noteworthy examples vulnerable to this attack are a social networking app and an online services client app. Both apps use the webkit view to enhance either the social networking experience or use online services (search, mail, etc.) and have 1.5 to 6 million installs. The two apps start the connection with a HTTP landing page, and we could rewrite the HTTPS redirects to HTTP and thus catch the login credentials for Facebook, Yahoo and Google.

One way to overcome this kind of vulnerability is to force the use of HTTPS, as proposed by the HTTP Strict Transport Security IETF Draft[11], or using a tool such as HTTPS-Everywhere.[12] However, these options currently do not exist for Android. Android's default browser as well as available alternatives such as Chrome, Firefox, Opera or the Dolphin Browser do not provide HTTPS-Everywhere-like features out of the box, nor could we find any add-ons for such a feature.

Lazy TLS Use

Although the Android SDK does not support TLS pinning out of the box, Android apps can also take advantage of the fact that they can customize the way TLS validation is implemented. Unlike general purpose web browsers that need to be able to connect to any number of sites as ordained by the user, many Android apps focus on a limited number of hosts picked by the app developer: for example, the PayPal app's main interaction is with paypal.com and its sister sites. In such a case, it would be feasible to implement TLS pinning, either selecting the small number of CAs actually used to sign the sites or even pin the precise certificates. This prevents rogue or compromised CAs from mounting MITM attacks against the app. To implement TLS pinning, an app can use its own `KeyStore` of trusted root CA certificates or implement a `TrustManager` that only trusts specific public key fingerprints.

To investigate the usage of TLS pinning, we cherry-picked 20 high profile apps that were not prone to the previous MITM attacks and manually audited them. We installed our own root CA certificate on the phone and set up an TLS MITM proxy that automatically created CA-signed certificates for the hosts an app connects to. Then, we executed MITM attacks against the apps. Table 6.3 shows the results. Only 2 of the apps make use of TLS pinning and thus were safe from our attack. All other apps trust all root CA signatures, as long as they are part of Android's trust anchors, and thus were vulnerable to the executed attack.

Missing Feedback

When an app accesses the Internet and sends or receives data, the Android OS does not provide any visual feedback to the user whether or not the underlying communication channel is secure. The apps are also not required to signal this themselves and there is nothing stopping an app from displaying wrong, misguided or simply no information. We found several apps that provided TLS options in their

[11]cf. `http://android-ssl.org/s/10` – last access 13.04.2016
[12]cf. `https://www.eff.org/https-everywhere` – last access 13.04.2016

App	Installs	TLS Pinning
Amazon MP3	10-50 million	
Chrome	0.5-1 million	
Dolphin Browser HD	10-50 million	
Dropbox	10-50 million	
Ebay	10-50 million	
Expedia Bookings	0.5-1 million	
Facebook Messenger	10-50 million	
Facebook	100-500 million	
Foursquare	5-10 million	
GMail	100-500 million	
Google Play Market	All Phones	
Google+	10-50 million	
Hotmail	5-10 million	
Instagram	5-10 million	
OfficeSuite Pro 6	1-5 million	
PayPal	1-5 million	
Twitter	50-100 million	✓
Voxer Walkie Talkie	10-50 million	✓
Yahoo! Messenger	10-50 million	
Yahoo! Mail	10-50 million	

Table 6.3: Results of the TLS pinning analysis.

settings or displayed visual security indicators but failed to establish secure TLS channels for different reasons.

We found banking apps in this category that we could not fully test, since we did not have access to the required bank accounts. However, these apps stated that they were using TLS-secured connections and displayed green visual security indicators, but suffered from one of the MITM attack vulnerabilities shown above. We were therefore able to intercept login credentials, which would enable us to disable banking cards and gather account information using the app.

We found several prominent mail apps that had issues with missing feedback. Both were dedicated apps for specific online services. The first app with an install base between 10 and 50 million users handled registration and login via a secure TLS connection, but the default settings for sending and receiving email are set to HTTP. They can be changed by the user, but the user needs to stumble upon this possibility first. Meanwhile, there was no indication that the emails were not protected.

An instant messaging app with an install base of 100,000 to 500,000 users transfers login credentials via a non-TLS protected channel. Although the user's password is transferred in encrypted form, it does not vary between different logins, so Eve can record the password and could use it in a replay attack to hijack the user's account.

We found a framework that provides a graphical app builder, allowing users to easily create apps for Android and other mobile platforms. Apps created with this framework can load code from remote servers by using the `dalvik.system.DexClassLoader`. Downloading remote code is handled via plain HTTP. We analyzed one app built with the framework and could inject and execute arbitrary Java code, since the downloaded code is not verified before execution.

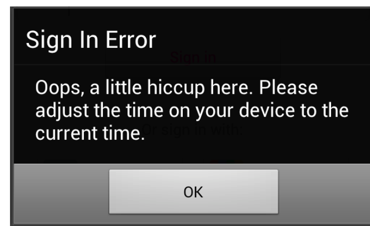


Figure 6.2: A sample warning message that occurs in an app that is MITM attacked.

During manual analysis, we also found that 53 apps that were not vulnerable to our MITM attacks did not display a meaningful warning messages to the user under attack. These apps simply refused to work and mostly stated that there were technical or connectivity problems and advised the user to try to reconnect later. There was also an app that recommended an app-update to eliminate the network connection errors. Some apps simply crashed without any announcement. Figure 6.2 shows a confusing sample error message displayed during a MITM attack.

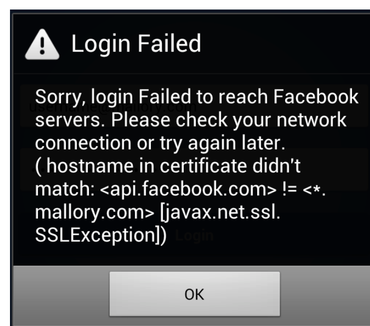


Figure 6.3: Facebook's TLS warning.

An additional 6 apps not vulnerable to our MITM attacks did display certificate related warning messages, but did not indicate the potential presence of a MITM attack. The official Facebook app is not vulnerable to the MITM attacks described above and is a positive example for displaying a meaningful warning message. Even if the warning message contains tech-savvy wording, the user at least has the chance to realize that a MITM attack might be occuring (cf. Fig. 6.3).

Interestingly – apart from browser apps – there was only one app that allows the user to choose to continue in the presence of an TLS error.

Limitations

This study has the following limitations: a) During static code analysis, the studied applications were selected with a bias towards popular apps; b) The provided install base numbers are only approximate values as provided by Google's Play Market; c) We only checked 100 of the apps where MalloDroid found occurrences of broken TLS implementations manually. For the rest, the existence of the unsafe code does not mean that these apps must be vulnerable to a MITM attack; d) Static code analysis

might have failed in some apps, for instance if they were obfuscated. Hence, there might be further vulnerable apps that we did not classify as such; e) During manual audits, the applications were selected with a bias towards popularity and assumed sensitivity of data they handle; f) We could not test the entire workflow of all apps, e. g., it was not possible to create a foreign bank account to see what happens after successfully logging into the bank account.

6.4 Userstudy: TLS Warning Messages

The previous sections in this chapter focused on developer issues implementing TLS certificate verification in Android apps and their impact on end users. However, this section goes beyond and presents a userstudy on the challenges for end users in case correct certificate verification is implemented and meaningful warning messages in error cases are presented.

The default Android browser is exemplary in its TLS use and uses sensible trust managers and host name verifiers. Also, unlike most special purpose apps, it displays a meaningful error message when faced with an incorrect certificate and allows the user to continue on to the site if (s)he wants to. Thus, it relies on the ability of the user to understand what the displayed warning messages mean and what the safest behavior is (cf. Figure 6.4). There have been many studies of this issue conducted in the context of desktop browsing. Here, to the best of our knowledge, we present the first survey to investigate the users' perceptions when using secure connections in the Android browser.

Online Survey

The goal of our online survey was to explore whether or not the user can assess the security of a connection in the Android browser. We wanted to test that a) a user can distinguish a HTTPS connection from a regular HTTP connection and b) how the user perceives an TLS warning message. Previous work has addressed the effectiveness of warning dialogues in several scenarios, mostly for phishing on regular computers (e. g., [60, 188]). Felt et al. [84] conducted a survey on the prompts informing users of the requested permissions of Android apps during installation. The online survey in this work is based on a similar design, but studies TLS certificate warnings and visual security indicators in Android's default browser.

Participants were recruited through mailing lists of several universities, companies and government agencies. The study invitation offered a chance to win a 600$ voucher from Amazon for participation in an online survey about Android smartphone usage. The survey could only be accessed directly from an Android phone. We served the survey via HTTPS for one half of the participants and via HTTP for the other. After accessing a landing page, we showed the participants a typical Android certificate warning message, mimicking the behavior of the Android browser. Subsequently, we asked whether the participants had seen this warning before, if they had completely read its text and how much risk they felt they are warned against. We also wanted to know whether or not they believed to be using a

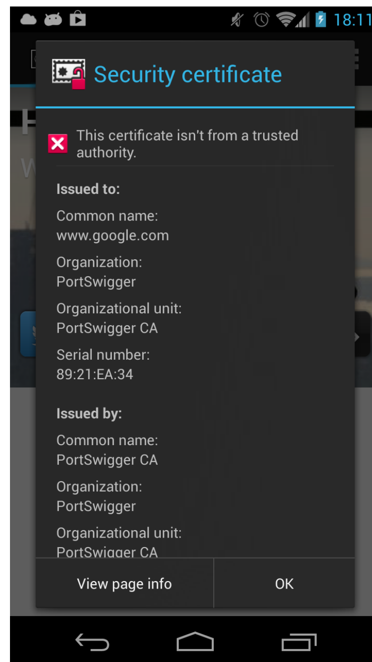


Figure 6.4: TLS warning message in Android's stock browser for Android versions earlier than 4.0.

secure connection and their reasons for this belief. Finally, we collected demographic information on technical experience, Android usage, previous experience with compromised credentials or accounts as well as age, gender and occupation. More online survey related information can be found in Appendix C.1, Page 198.

Results

754 participants completed the survey. The average age was 24 years ($sd = 4.01$), 88.3% were students while the rest mainly were employees. 61.9% of our participants did not have an IT-related education or job (non-IT experts in the following) and 23.2% had previous experience with compromised credentials or accounts. Overall, the self-reported technical confidence was high: participants stated a mean value of 4.36 for IT experts and 3.58 for non-experts on a scale from 1 (often asking for help) to 5 (often providing help to others). 51.9% of IT experts and 32.8% of non-IT experts have been using an Android smartphone for more than a year and 57.1% of experts and 69.8% of non-experts had only 25 apps or less installed.

Concerning connection security, we found that 47.5% of non-IT experts believed to be using a secure connection, while the survey was served over HTTP. On top of that, even 34.7% of participants with prior IT education thought that they were using a secure channel when they were not. In both groups, 22.4% were unsure about the protection of their connection. Only 58.9% of experts and 44.3% of non-experts correctly identified that they were using a secure or insecure connection when

prompted. The majority of users referred to the URL prefix as the reason for their beliefs and 66.5% of participants that were unsure said that they did not know how to judge the connection security. Those users that were wrongly assuming a secure connection stated that they use a trustworthy provider (47.7%), trust their phone (22.7%) or thought that the address was beginning with https:// even though it was not (21.6%) as a justification for their beliefs. Interestingly, participants that stated that they had suffered from compromised credentials or online accounts before did significantly better in judging the connection state ($\chi^2 = 85.36, df = 6, p < 0.05$).

Concerning the warning message, the majority of participants stated that they had not seen such a certificate warning before (57.6% of non-IT experts and 52.3% of IT experts) or were unsure (5.9%/9.2%). 24% of all participants only read the warning partially and 4.5% did not read it at all. These numbers did not differ significantly based on whether or not they had seen the warning before. The participants rated the risk they were warned against with 2.86 ($sd = .94$), with 1 being a very low risk and 5 a very high risk. The perceived risk did not differ significantly between IT-experts and other users.

Overall, the results of our online survey show that assessing the security of a browser session on Android's default browser was problematic for a large number of our participants. While certificate handling is done correctly by the browser app and basic visual security indicators are offered, the user's awareness for whether or not her or his data is effectively protected is frequently incomplete.

Limitations

Our survey is limited in the following ways: We used official mailing lists to distribute the invitation for the survey. While, on a technical level, this should not affect the trustworthiness of the mail or the survey site - we did not digitally sign the emails and we served the survey with a URL that was not obviously linked to the university. Therefore, the emails could have been spoofed. Nonetheless, it is likely that a higher level of trust was induced in most participants, due to the fact that the survey was advertised as a university study (c.f. [183]). We therefore refrained from evaluating the users' reasons for accepting or rejecting a certificate in this concrete scenario.

Participants were self-recruited from multiple sources, but we received mainly entries from university students for this first exploration. While a study by Sotirakopoulos et al. [184] found little differences between groups of students and the broader population in the usable security context, a more varied sample of participants would improve the general applicability of the results.

6.5 Summary

In this work, we presented an investigation of the current state of TLS usage in Android and the security threats posed by benign Android apps that communicate over the Internet using TLS. We have built MalloDroid, a tool that uses static code analysis to detect apps that potentially use TLS inadequately or incorrectly

and thus are potentially vulnerable to MITM attacks. Our analysis of the 13,500 most popular free apps from the Google Play Market has shown that 1,074 apps contain code belonging to this category. These 1,074 apps represent 17% of the apps that contain HTTPS URLs. To evaluate the real threat of such potential vulnerabilities, we have manually mounted MITM attacks against 100 selected apps from that set. This manual audit has revealed widespread and serious vulnerabilities. We have captured credentials for American Express, Diners Club, Paypal, Facebook, Twitter, Google, Yahoo, Microsoft Live ID, Box, WordPress, IBM Sametime, remote servers, bank accounts and email accounts. We have successfully manipulated virus signatures downloaded via the automatic update functionality of an anti-virus app to neutralize the protection or even to remove arbitrary apps, including the anti-virus program itself. It was possible to remotely inject and execute code in an app created by a vulnerable app-building framework. The cumulative number of installs of apps with confirmed vulnerabilities against MITM attacks is between 39.5 and 185 million users, according to Google's Play Market.

The results of our online survey with 754 participants showed that there is some confusion among Android users as to which security indicators are indicative of a secure connection, and about half of the participants could not judge the security state of a browser session correctly. We discussed possible countermeasures that could alleviate the problems of unencrypted traffic and TLS misuse. We offer MalloDroid as a first countermeasure to possibly identify potentially vulnerable apps.

This chapter illustrated the challenges for developers and end users of Android apps when using TLS. On the one hand, our end user study confirmed findings of previous works for desktop computers: Users are overwhelmed when confronted with security warnings and often fail to react securely. On the other hand, identifying developer issues when dealing with custom certificate validation revealed that usability also is of great importance when developing secure software leading me to an important assumption: Similarly to end users and administrators, developers are limited by security decisions made by other actors in the ecosystem. In case of TLS certificate validation, a careful developer can increase the security and privacy for her end users drastically by using a valid TLS certificate to secure network connections. However, to successfully and securely implement customized TLS certificate validation, developers need a thorough understanding of TLS, i. e., a strong security background. This chapter highlights the possible impact of an ecosystem that is designed with consideration for the usability of security relevant mechanisms for software developers.

Therefore, the next chapter investigates the root causes of why many developers fail when implementing customized certificate validation. Based on these results, I propose an easy to use redesign of how TLS development can be implemented in the future.

7 System Designers: Rethinking TLS Development

Disclaimer: *The contents of this chapter were previously published as part of the paper "Rethinking SSL Development in an Appified World" presented at* 20th *ACM Conference on Computer and Communications Security (CCS) in 2013 [81] together with co-authors Marian Harbach, Markus Kötter, Henning Perl and Matthew Smith (alphabetical order). As this work was conducted with my co-authors, as a team, this chapter will use the academic "we" to mirror this fact. The idea and initial concept for this work came from myself. I developed the system design for the integration of a novel TLS architecture into the Android operating system. I manually analyzed iOS apps for their correct TLS certificate validation usage. The interviews conducted with developers of vulnerable mobile apps were designed, conducted and analyzed by myself. However, my co-authors contributed in different ways. Implementing the integration of our novel TLS architecture into the Android operating system was joint work with Markus Kötter. Together with Henning Perl, Marian Harbach and Matthew Smith, we jointly discussed the work's implications and compiled the paper for publication.*
Software described in this chapter is available at `https://zenodo.org/record/51120`.

7.1 Motivation

In the previous chapter, an in-depth study of 13,500 Android apps showed that a large number of apps did not use TLS correctly, thus making them vulnerable to MITM attacks. The affected applications ranged from home-brew and open source apps to those developed by large corporations and security specialists, suggesting that TLS problems are not just a matter of untrained developers getting it wrong.

While the previous chapter and a related study by Georgiev et al. [96] discuss possible reasons for why so many apps across such a wide range of developers are affected and make recommendations on how to prevent these problems in the future, the actual causes of the problems have not yet been identified nor have the potential countermeasures been evaluated. In this work, we continue where this work left off and evaluate the root causes of TLS coding problems. Based on these findings, we argue that the way developers work with TLS needs to be changed significantly. We designed a framework for TLS development to demonstrate our proposal, implemented it for Android, evaluated it against the set of 13,500 popular apps we analyzed in our previous study and conducted developer interviews to show its functionality and feasibility.

Our solution makes several changes to the status quo: First, it removes the need for developers to write actual TLS code. Instead an app developer can turn on and configure TLS using only configuration options in case they intend to deviate from the standard use case. Second, unlike in the current system, all use-cases we found in apps and those described by developers are supported securely, removing the need for dangerous customization. Third, the framework can make a distinction between developer devices and end user devices. This allows developers to rapidly prototype applications and tinker without much effort, but it also automatically

and properly protects end users. Fourth, in the remaining rare cases that there is a problem with TLS, the end user is reliably informed about the nature of the problem which means that apps can no longer silently ignore these warnings and invisibly make users vulnerable to MITM attacks. Finally, our approach allows for novel TLS validation strategies and infrastructures, such as Certificate Transparency [136], Convergence [146], and AKI [128], to be deployed centrally instead of potentially requiring tens of thousands of app developers to make adjustments to their code, thus significantly easing the deployment of novel TLS validation strategies[1].

Our contributions can be summarized as follows:

- We conducted the first analysis of iOS TLS security to ascertain whether the walled garden approach and stricter code auditing process of Apple could prevent the kind of problems previously seen on Android. We manually examined 1,009 iOS apps. The results show that, similar to Android, 14% of apps using TLS do not implement TLS validation correctly and are thus vulnerable to active MITM attacks.
- We conducted the first in-depth study of the reasons behind the widespread problems with TLS on the two major app platforms Android and iOS, including both technical aspects as well as an in-depth study with affected developers.
- Based on the above research, we designed and implemented countermeasures for handling TLS on Android. This solution can also serve as a blueprint for TLS handling on other appified platforms.
- We conducted an extensive evaluation of our approach by auditing 13,500 apps and showing that our solution covers all use-cases present in these apps. We also conducted a follow-up developer study to ensure our approach does not break any development needs and would find acceptance within the target community.

7.2 Background

TLS on Appified Platforms

Background information on TLS can be found in Chapter 2.3.3 on Page 27.

Research has shown that the appification and app market trend has caused new security and privacy challenges for users, developers and researchers [174, 49, 63, 210, 76, 34, 196, 159, 84]. This research focused mainly on the threats posed by malicious entities and their apps. The problems with TLS we [76] and Georgiev et al. [96] discovered are different in that they are not caused by malicious intent, but nonetheless pose a serious threat.

While the bulk of TLS connections on desktop systems occurs in browsers which validate TLS certificates correctly, there are also applications that use TLS to protect

[1]We believe this last feature could be useful in the light of the many novel solutions (e. g. [136, 146, 128]) suggested in this domain that are currently facing adoption/deployment problems.

their communication. Georgiev et al. [96] analyzed the security of TLS certificate validation in a wide range of TLS libraries and programming frameworks. They conclude that many popular libraries fail when it comes to TLS certificate validation and thus endanger the applications which are based on these libraries.

In Chapter 6 we examined the state of TLS on Android. We analyzed the TLS security of 13,500 popular free apps from Google's Play Market. The results showed that 1,074 apps contained TLS code that either accepted all certificates or all hostnames for a certificate and thus leaves the users potentially vulnerable to MITM attacks. The cumulative install base of the apps with confirmed vulnerabilities against MITM attacks ranged between 40 and 185 million users.

Georgiev et al. [96] make recommendations for what could be done to alleviate these widespread problems. They recommend that app developers should use fuzzing and adversarial testing. Developers should not modify application code and disable certificate validation for testing with self-signed and/or untrusted certificates, but create a temporary keystore with the untrusted CA's public key in it. They also recommend that app developers should not rely on TLS libraries to do things correctly for them. Instead they should set all necessary security parameters themselves explicitly. For library developers, they recommend that TLS libraries be made more explicit about the semantics of their APIs. They also recommend that libraries use safe defaults as much as possible. Furthermore, they should not silently skip important functionalities such as hostname verification and should remain consistent in using return values or flags for reporting purposes.

Our central suggestion was to drastically limit custom TLS handling in apps. In an alternative approach we suggested that a static code analysis be performed by the app store or the app installer to then inform developers/users about potentially unsafe code.

Both works only briefly describe their potential countermeasures as part of their recommendations for future work. No implementations or evaluations were presented. Interestingly, the countermeasures suggested by the two parties diverge somewhat in their direction. Georgiev et al. call for better APIs for developers, emphasize that developers need to check their apps themselves instead of relying on libraries to do things correctly and to work around API restrictions using custom keystores instead of modifying validation for development purposes, while we suggest almost the opposite approach by recommending to limit the developers' capabilities for customizing TLS and adding checks to prevent broken apps from entering the market or the device. Subsequently, we will examine the root causes of the problems facing developers when using TLS to better judge which countermeasures are likely to achieve the best results.

TLS Development Paradigm

There are countless TLS libraries and TrustManagers that aim to make the integration of TLS into apps easier. However, as the studies by Georgiev et al. [96] and us [76] have shown, many of these are either broken or so error-prone that developers break their apps using them. This work makes important contributions to both

these areas.

7.3 TLS on iOS

To examine whether or not the widespread TLS problems of apps and libraries introduced above are endemic to the Android and open source ecosystems, we conducted the first in-depth analysis of iOS apps to see if the more restrictive and curated Apple App Store would prevent apps with broken TLS code from entering the iOS ecosystem. While iOS does not offer developers as many options as Android, it still offers similar freedom concerning the implementation and use of TLS. Developers can decide if they want to use TLS or not and just like on Android they can use TLS but can turn TLS certificate validation off. They are also left alone with the challenge to find an appropriate way to handle certification validation errors and inform the user: In terms of TLS APIs, Android and iOS are fairly similar. However, Apple performs a code analysis on all apps, in order to prevent apps that do not conform to their policies from entering the store.

We did not have the means to automatically crawl Apple's app store and use static code analysis on tens of thousands of apps as we did with Android, so we opted for a manual approach. Initially, we downloaded 150 cherry-picked apps for analysis. Based on the findings in these apps we conducted developer interviews as described in section 7.4. We then extended our study to include 1,009 apps for a more robust evaluation. We downloaded the 1,009 apps by selecting the most popular free apps. Since iOS does not work with permissions the way Android does, it was not possible to see which apps have access to sensitive information and can connect to the Internet before installation. We therefore installed all of the apps on an iPhone 4S running iOS 5 to study them in action. We then mounted active Man-In-The-Middle attacks against TLS connections using a transparent proxy to see how the apps react and what kind of sensitive information we could gather.

We captured network traffic from 884 apps and TLS-protected network traffic from 697 of these 884 apps. Of these 697 apps, 98 (14%, 9.7% of all 1,009 apps) were vulnerable to our MITM attack and leaked sensitive information. Of the remaining 599 apps that were not vulnerable to our active MITM attack, 312 apps presented the user with a warning message; 58 apps presented a warning message indicating that there were problems with the TLS certificate. 254 presented warning messages that did not give an appropriate description of what was going on, for instance stating that the login credentials were wrong or that there was no Internet connection available. Finally, 287 apps simply did not connect to the attacked host either doing nothing, hanging indefinitely or crashing. Additionally, 82 (9.27%) of the 884 apps used plain HTTP connections to transfer sensitive information. Two apps were vulnerable to TLS stripping attacks. One of these was an online banking app that loaded the bank's website via plain HTTP, while the other app connected to the HTTP version of Facebook. Thus we were able to gather sensitive information from 182 apps, i. e. 20.5% of apps from which we captured network traffic. The information included the usual suspects of login credentials, banking accounts, data

stored on cloud storage services, emails, or chat messages.

This shows that the TLS problems on iOS are similar to those on Android and that Apple's more restrictive and curated app development model and App Store do not prevent TLS-related security issues. Just like on Android, app developers turn off TLS certificate validation, write apps that are vulnerable to TLS stripping and even in cases where apps apply TLS certificate validation correctly, they often do not present sensible feedback when validation fails.

While there are many more details worth discussing in the context of iOS, for the purpose of this work, the fact that iOS apps suffer from a similar number of TLS problems shows that these problems must have underlying causes which are not specific to a platform or app store model.

For instance, when presented with an invalid certificate, the official Facebook iOS app only shows a sad smiley, that says *"something went wrong"* and suggests the user to try again (cf. Figure 7.1). So as on Android many app developers leave their users entirely open to attack and even those who don't, do not warn them sufficiently when an attack does occur.

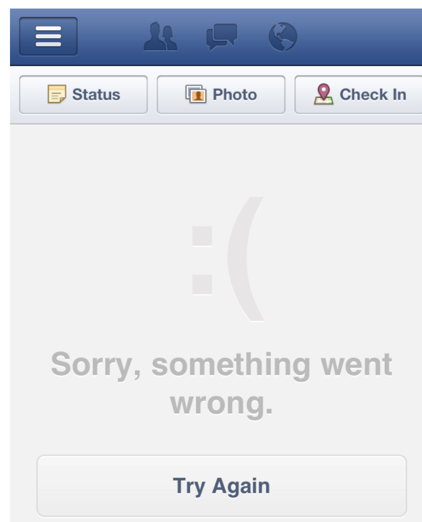


Figure 7.1: The iPhone Facebook app shows this warning message in case TLS certificate validation fails.

iOS Development Frameworks

Several of the vulnerable apps we found were created using popular programming frameworks. Since any bug introduced by such a framework could potentially affect a large number of apps, we decided to take a closer look at these frameworks. During our analysis, we identified two cross-platform mobile application SDKs and an iOS networking wrapper library that all create code that contains vulnerable TLS certificate validation.

MKNetworkKit The MKNetworkKit[2] is a networking wrapper library for iOS with the aim to be easy to use and to simplify the iOS default networking stack. In online forums, such as `stackoverflow.com`, developers often complain about problems with iOS's built-in networking APIs, accounting for the popularity of libraries such as MKNetworkKit, which provides lightweight methods for standard networking tasks. The library can also handle HTTPS requests, but fails when it comes to TLS server certificate validation. In the `MKNetworkOperation.m` module, custom certificate validation is implemented. Instead of throwing an exception when certificate validation fails, the code silently continues as if no error occurred. The library is also available for Mac OS X., exhibiting the same problems there as well. Thus, iOS apps using the MKNetworkKit library for networking tasks are vulnerable to active MITM attacks.

Titanium Framework The Titanium cross platform mobile application SDK[3] is a JavaScript-based platform which enables developers to write mobile apps in JavaScript and automatically translates them into native mobile apps. Titanium targets iOS, Android and HTML5, making it particularly attractive for web developers who want to create mobile apps. While the Titanium framework generates secure TLS code for Android, TLS certificate validation for iOS apps is turned off. Based on app creation statistics posted on their website, this could affect more than 30,000 apps built with the framework. During our manual app testing, we found a car sharing app (cf. Zipcar `http://www.zipcar.com`) that was built with the Titanium framework and is promoted as a flagship app on the Titanium website. The app belongs to the world's largest car sharing service and is available in North America and Canada. In 2011, the app was labeled one of the 50 best iPhone apps by Time Magazine. Features of the app include locating and booking cars, sounding the horn and unlocking the doors. The app uses TLS to transmit login credentials but does not validate TLS certificates. Thus, an attacker can steal credentials and subsequently is able to book and unlock a car in the name of the attacked user.

PhoneGap PhoneGap[4] is a free open source framework for developing mobile apps for seven platforms including Android and iOS, using HTML, CSS and JavaScript. It contains dedicated classes for data transfer that include customized TLS verification code. For Android and iOS, the framework produces code that effectively turns TLS certificate validation off. If developers do not manually check the generated code, they will not see the comment and thus not be aware of the problem. According to PhoneGap[5], more than 23,000 apps could be affected.

[2] `http://blog.mugunthkumar.com/products/ios-framework-introducing-mknetworkkit/` – last access 13.04.2016

[3] `http://www.appcelerator.com/platform/titanium-sdk/` – last access 13.04.2016

[4] `http://phonegap.com/` – last access 13.04.2016

[5] `http://www.slideshare.net/AndreCharland/phone-gap-stats-growth` – last access 13.04.2016

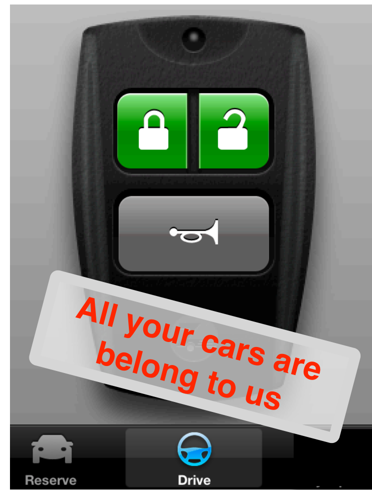


Figure 7.2: Users of the car sharing App can unlock their cars with the app – and so can attackers.

7.4 Developer Study

The iOS study shows that the trouble developers have using TLS correctly is common to the major platforms and across all applications and platform paradigms. To develop an effective countermeasure, we wanted to identify the root causes of these problems. Therefore, we studied entries about TLS development for both Android and iOS in online forums and conducted interviews with developers who had produced broken TLS code.

Online Forums

We used `stackoverflow.com`, a popular online forum for software developers, to search for results that contained text such as *“android/ios allow all certificates”* and *“android/ios trust all certificates”* first. These kinds of results were found in threads where developers asked how they could *“work with self-signed certificates”* or *“make 'javax.net.ssl.SSLException' go away”*. Answers to these and similar questions mainly contained explanations on how to turn off TLS certificate validation or hostname verification without mentioning that this would create serious security issues. Studying these discussions confirmed our impression that many developers on both platforms – iOS as well as Android – lack an adequate understanding of how TLS works and were frustrated with the complexity of customizing TLS code and thus willing to use the quick fixes offered by the forums – potentially without understanding the consequences. These threads had more than 30,000 views. To put this number into perspective, searching for *“android ssl pinning”* or *“ios ssl pinning”* returns only one result for Android and two results for iOS with less than 600 views in total.

Interviews

While online forums give a good indication of potential issues, a more reliable way to confirm whether or not these issues are the actual reasons causing the problems in the thousands of real world apps is to talk to developers. We contacted 78 developers of the 82 vulnerable apps[6] from the 100 Android and 150 iOS apps we studied in detail. We informed the developers of the discovered vulnerabilities via email and kindly asked them to contact us for further information and assistance in fixing their problems. We received responses from 39 of the 78 developers. We disclosed the respective vulnerability, offered further assistance and asked whether the developers were willing to discuss the details of the security bug either via telephone or email. We promised the developers to anonymize all information they provided and to make neither their names nor the apps' names publicly available. 14 developers agreed to an interview while the rest was not willing to discuss the topic in further detail – often with regard to "constraints" dictated by their legal departments. The interviews were conducted in German or English, depending on the developer's origin. However, not all developers were native speakers. Statements were translated by the authors, grammatical errors were not corrected.

Ethical Considerations

Our university is located in Germany and thus our study was not required to pass an IRB review. Nevertheless, the interviews complied with the strict German privacy regulations. We do not disclose names of affected apps or developers and all information was evaluated anonymously. After the interviews, the participants were debriefed and any questions were answered.

Results

One of the main causes for the problems we found, was that developers wanted to use self-signed certificates during development. 5 of the 14 developers reported that they needed to turn off TLS certificate validation during development because they were working with test servers that used self-signed TLS certificates. To avoid certificate validation exceptions, they implemented their own TLS certificate validation strategies that accept all certificates, or copied code from the online forums mentioned above that promised to help with getting out of the *"self-signed certificate dilemma"*. While it is understandable that developers turn off TLS certificate validation in the development phase, these developers basically forgot to remove their accept-all code when they released their apps. Three of these five developers realized that this was a serious security threat and stated that they should fix this security issue as soon as possible. The other two did not see the problem and even after our explanations stated the following:

> *"You said that an attacker with access to the network traffic can see the data in cleartext. I tried that and I connected my phone to a Wi-Fi hotspot on my*

[6] Some developers were responsible for both an Android and an iOS app; thus there were only 78 developers for 82 apps.

laptop. When I used Wireshark to look at the traffic, Wireshark said that this is a proper TLS protected data stream and I could not see any cleartext information when I manually inspected the packets. So I really cannot see what the problem is here."

This supports the hypothesis stated by Georgiev et al. [96] that too little adversarial testing is conducted by app developers. However, it also raises the issue that there are developers, who, while being technically adept enough to use Wireshark to check if their app's traffic is really encrypted, do not understand the nature of the threat and thus take no precautions to counter it.

Apart from developers wanting to use self-signed certificates during development, we also talked to developers who actually wanted to use them in their production environment but were unaware of the security implications of accepting any certificate:

"I was using a self-signed certificate for my app because it is free and CA-signed certificates cost a lot. But, actually, I had no idea that working with self-signed certificates could have resulted in such a security issue. I think the online forum where I found the code snippet only said that it makes self-signed certificates work."

"We added this piece of code because our client uses an TLS certificate for his web-service which was signed by a certificate authority that is not pre-installed on Android and actually we did not realize that this would cause such trouble."

Sometimes, the broken TLS code was added because developers had difficulties understanding the problem and just went for the first solution that seemingly made the problem disappear:

"This app was one of our first mobile apps and when we noticed that there were problems with the TLS certificate, we just implemented the first working solution we found on the Internet. [...] We usually build Java backend software for large-scale web services."

However, there were also developers who even after being informed about the problems and the threat scenario did not properly understand the problem and their countermeasures did not address the threat arising from a MITM attack:

"We hadn't realized that it would cause such an issue by using self-signed certificates in the past time, and we just verified if the certificate was expired. But after noticing this issue, we strengthened the security check like verifying host name. We believe this improvement can ensure users' security. So we still stick to trust self-signed certificates right now for its smaller size and lower bandwidth cost."

So while they added hostname verification after we informed them about the issue, they still accept all self-signed certificates thus defeating hostname verification entirely. In another case, a development company of a vulnerable online banking app needed two iterations to fix their app correctly, even though we had sent them

the necessary code snippets. There were also cases where developers thought using broken TLS was adequate to protect information that they deemed to be not that valuable:

> *"We checked into this. Only the [...] feature is using a weak TLS certificate and that connection only sends the device models and IMEI, but that's not a security concern."*

Some developers knew that their code could cause security problems but saw no other option but to work with self-signed certificates by turning off certificate validation entirely, since their customers wanted to use self-signed certificates.

> *"This issue exists because many of our customers use self-signed certificates for SSO (single sign on). Some time back, a fix was implemented to allow this to work."*

One of those developers raised the interesting point that Android does not offer any default warning, forcing developers to provide one for themselves if they wish to inform users about failed certificate validations:

> *"The app accepts all TLS certificates because some users wanted to connect to their blogs with self-signed certs and [...] because Android does not provide an easy to use TLS certificate warning message, it was a lot easier to simply accept all self-signed certificates."*

The consequence of this design decision was that all users of this app were at risk because some wanted to use self-signed certificates.

Finally, the same developer accidentally left two connections unsecured:

> *"After you informed me about the TLS security issue in my app, I re-checked the code and another issue that exists came to my mind. I found two endpoints for RESTful web services I actually wanted to talk to via HTTPS, but actually I forget the s and now the app talks unencrypted HTTP. For users of an app and even for me as the developer it would be nice to have an indicator whether an app uses TLS for a network connection and even to have something that uses TLS by default for as many connections as possible."*

The iOS developers in our study tended to rely on frameworks and libraries during development and were understandably startled and upset when told that their apps were endangered because of faulty code generated by the framework (cf. Appendix 7.3, Page 116):

> *"I am using the MKNetworkToolkit as a network wrapping library and its TLS features for HTTPS. After you informed me of the issue I checked the library's code and found that by default TLS certificate validation is off. But, when I used the library in my app, I trusted it and did not check for the TLS MITM attack vulnerability because it is a widely used library."*

> *"When I was starting to build apps for iOS, I had a strong background in coding web applications. When I came up with the Titanium framework that allows*

> *developers to build native mobile apps by writing HTML and JavaScript, I decided to use this framework just because it was easier for me. [...] I never thought that the framework would produce broken code for TLS encryption and [...], although we conduct security audits for our apps, we did not include TLS certificate validation checks into the audit process."*

The feedback from app developers confirms that developers struggle to implement TLS correctly when they have a need to deviate from the standard use-case. They also rely on the implementations of frameworks and libraries to protect their apps without thoroughly testing either the frameworks' or their app's security. However, our investigation also provides some new insights: developers modified certificate validation code for internal testing purposes, e. g. working with arbitrary self-signed certificates on test servers, but forgot about that and thus did not remove the code for the production environment. So even those developers who understood the current need for signed and trusted certificates put their customers at risk. Also, the developers' problems did not only lie in the complexity of the code, but were based on a lack of understanding of how TLS works. There were also some cases where developers turned TLS validation off because of a customer's request, either accepting or not realizing the implications. Even when we explained what could go wrong and how to correct it, developers struggled when trying to fix their app (cf. Section 7.4). Altogether, our results imply that code-level customization of TLS-handling is an overwhelming problem for many developers and that there is a fairly high level of frustration with the complexity of adapting code to their use-cases.

Summary

After studying code snippets and advice in developer forums, as well as talking to app developers that use broken TLS certificate validation, we believe that in most cases when Android and iOS developers deviate from default TLS certificate validation strategies – that are secure on both platforms by default – they apply customization features in a way that weakens security significantly (c.f. Section 7.5 for a quantitative confirmation). Many developers of affected apps seem to have only a partial understanding of what TLS does and how it works. Yet, there are also developers who complain about the bad support for self-signed certificates and the lack of easy to use TLS warning messages.

One interesting aspect we found in the interviews was that developers in general seem to be interested in providing a high level of security for their users. We offered all developers to help them with their TLS problems and most of them took the offer. After giving background information on the security model of TLS and certificate validation, 10 of the 14 interviewed developers accepted our assistance. 7 of these 10 developers decided to strengthen their app's security by implementing TLS pinning. They did this because it gave them full control over the TLS certificates trusted by their app and they found that this was the most secure, flexible and cheapest way to provide a high level of security. We provided the developers with code to integrate

TLS pinning based on Moxie Marlinspike's GitHub page[7]. All developers agreed that being able to control exactly which certificates their apps trust is a great way to increase security, but that they would not have known how to do this without our help.

Our results imply that allowing app developers to customize TLS handling on source-code level overburdens many developers and leads to insecure apps. While it is easy to weaken app security by removing the default TLS certificate validation code, it is hard to strengthen it by implementing pinning or other security-strengthening strategies. Only one developer stated that using insecure TLS should not be taken too seriously, which makes us believe that in most cases insecure TLS connections are unintentional and that users must be protected against careless developers and developers who usually are no security experts.

Follow-up Analysis

Our developer study showed that many developers were unaware of the dangers facing their TLS connections and interested in fixing the issues. Some of the developers who accepted our help were capable of fixing the TLS problems in their apps. To analyze how developers cope with this situation without direct help, we ran a follow-up analysis. All affected developers (iOS & Android) were informed about the vulnerabilities and the possible security consequences for their users at the time of discovery of the vulnerability. They were given the recommendation that they should fix the identified issues as soon as possible. Three months after the respective notifications, we downloaded the apps again to check if they had fixed the TLS vulnerabilities. We found that 51 of the 78 developers did not fix the TLS issues. Six apps were not available any more which means we could not test them a second time. Only 21 (26.9%) apps were fixed and implemented correct TLS certificate validation in their current versions. Of these 21 apps, 9 belonged to developers we had helped directly during the interview process. However, 5 of the 14 developers we interviewed did not fix the TLS issues in their apps. Furthermore, developers of an app that included vulnerable TLS certificate validation on both Android and iOS only fixed it for Android while the iOS app was still vulnerable in the second test.

The results of this follow-up analysis indicate that even after informing and educating developers about vulnerabilities in their TLS code, problems in correcting these mistakes and deploying a safe solution still remain. Finding that 73.1% of all informed developers did not fix the reported TLS issues demonstrates even more that the current TLS mechanisms on appified platforms need rethinking.

7.5 TLS Development Re-thought

In the previous sections, we showed that incorrect TLS validation is a widespread problem on appified platforms and analyzed the causes of these issues. In a follow-up

[7] `https://github.com/moxie0/AndroidPinning` – last access 13.04.2016

study, we found that only a small part of previously vulnerable apps had fixed their app's TLS vulnerabilities, even after we informed the developers about the problems. In the following, we propose a major change in how system designer could help app developers to address these issues. While we implemented our ideas for the Android platform, they can serve as a blueprint for system designers to increase TLS security on other platforms such as iOS, but also OS X and Windows 8 which are moving towards the app paradigm as well.

Based on our analysis described above, we believe that simplifying TLS libraries or trying to educate developers in the context of TLS security will not solve the problem. For most developers, network security is not a primary concern and they just want to "make it work". An ideal solution would enable developers to use TLS correctly without coding effort and prevent them from breaking TLS validation through customization. However, it is also important not to restrict their capabilities to produce functional (and secure) applications: If our solution does not offer the needed functionality, developers will be tempted to break it just like they are breaking the built-in TLS code at the moment.

Before going into detail, we summarize the desired features for TLS validation identified during analysis:

Self-Signed Certificates – Development. Developers commonly wish to use self-signed certificates for testing purposes and hence want to turn off certificate validation during testing.

Self-Signed Certificates – Production. A few developers wanted to use self-signed certificates in their production app for cost and effort reasons.

Certificate Pinning / Trusted Roots. Developers liked the idea of having an easy way to limit the number of trusted certificates and/or certificate authorities.

Global Warning Message. Developers requested global TLS warning messages since they described building their own warning messages as too challenging.

Code Complexity. Developers described the code-level customization features of TLS as too complex and requiring too much effort.

In addition to these developer requirements, we add a user requirement to our list of desired changes on how TLS is handled in apps. We base this goal on the related area of TLS handling in browsers. While a website can choose not to offer TLS at all, it cannot prevent the browser from warning the user about an unsafe connection. The website also cannot turn off TLS validation for the user. In the world of apps, developers currently have the power to define TLS policies for an app without those being transparent for the user. While there are well known usability problems with TLS warning messages in browsers [188], we believe that allowing developers to for instance silently ignore TLS errors and put the user at risk is worse. We thus define an additional goal:

User Protection. The capabilities of a developer should be limited in a way that prevents them from invisibly putting user information at risk.

To achieve all these features, several changes to the way TLS is used on appified platforms are necessary. First and foremost, we propose the following paradigm-shift: instead of letting all developers implement their own TLS code (and potentially break TLS in the process, with no chance for the user to notice), the main TLS usage patterns should be provided by the OS as a service that can be added to apps via configuration instead of implementation. This is a fairly radical shift in responsibility, however, we believe there now is enough evidence to warrant this move. Furthermore, the evaluation of our system presented in Section 7.5 shows that it is both technically possible and acceptable from the developer's standpoint. Configuration instead of implementation also lends itself well to offer other requested features, such as allowing developers to turn off TLS certificate validation for their app on their device in the settings during development. This would allow the use of self-signed certificates during development, but would not affect the installation of an app on a user's device. Surprisingly, none of the major mobile or desktop operating systems provide this feature, although we believe it would offer significant benefits to all parties.

The platform should offer configurable options for the new TLS service so that developers cannot and need not circumvent security features on the code level. The simple removal of the need to tinker with TLS security aspects for testing purposes will already reduce the amount of vulnerable apps significantly. It will also protect users from developers who do not understand how TLS works and who therefore make honest mistakes during implementation.

Table 7.1 gives an overview of our proposed modifications compared to the traditional code-level approach.

Implementation on Android

Figure 7.3 gives a high-level overview of the modifications we implemented to create the proposed TLS service on Android. The white boxes contain classes we modified or created for our solution. The dashed lines show Android components that are now circumvented since they proved to be insecure. The grey boxes are comments on what the different components do. The start arrow shows the entry point where app code passes control over to the central TLS system. The features offered by our solution are presented in the following sections.

Features

Mandatory TLS Validation

As stated above, we propose that the capability and need to customize TLS certificate validation and hostname verification on source code-level is removed. Instead, TLS certificate validation should be enforced for every TLS handshake automatically, while taking into account the different usage scenarios such as development vs. production.

To this end, we provide the `TrustManagerClient` and `TrustManagerService` that replace the capabilities of Android's default TrustManager (cf. Figure 7.3). We

	CA Validation	CA Pinning	Certificate Pinning	Development Mode	Logging	Validation Strategies
Standard	✓	—	—	—	—	—
Our approach	✓	✓	✓	✓	✓	*P*

Table 7.1: A comparison between the status quo and our approach concerning validation features.
✓ = supported out of the box;
− = custom code required;
P = pluggable.

only modify methods which are private and final, thus binary compatibility is given and we do not break modularity. More information on the compatibility of our approach can be found in Section 7.5. Both the client and service part of our TLS validation implementation prevent Android apps from using broken certificate validation. Upon creation of a socket, the newly developed `TrustManagerClient` automatically requests TLS certificate validation from the service counterpart. App developers cannot circumvent secure validation anymore, since customized TrustManager implementations are prevented by our modification. The `TrustManagerService` enforces TLS certificate validation against the trusted root CAs and can drop the connection or present the user with a warning message in case validation fails (more on this in Section 7.5).

To mandate secure hostname verification, we patched all stock hostname verifiers to enforce browser compatible hostname verification. We also added hostname verification to the central `SSLSocketFactory` (cf. Figure 7.3). Hostname verification is conventionally delegated to the application layer: With HTTPS for example, the hostname for verification is extracted from the requested URL. In contrast, Android's SSLSocketConnection implementation does not check the hostname, even though it may have been provided in the method call. Our patch improves this behavior by verifying hostnames with the parameters provided during connection establishment for any TLS connection.

This strict enforcement could cause developer issues in some usage scenarios described by our study participants, so several configuration options are described in the following in order to adapt our solution to different situations. Additionally, we discuss potential pathological cases in Appendix 7.5, Page 135.

Self-Signed Certificates

To allow developers to use self-signed certificates for testing purposes, we add a new option (cf. Figure 7.4) to the *Developer* settings, allowing app developers to

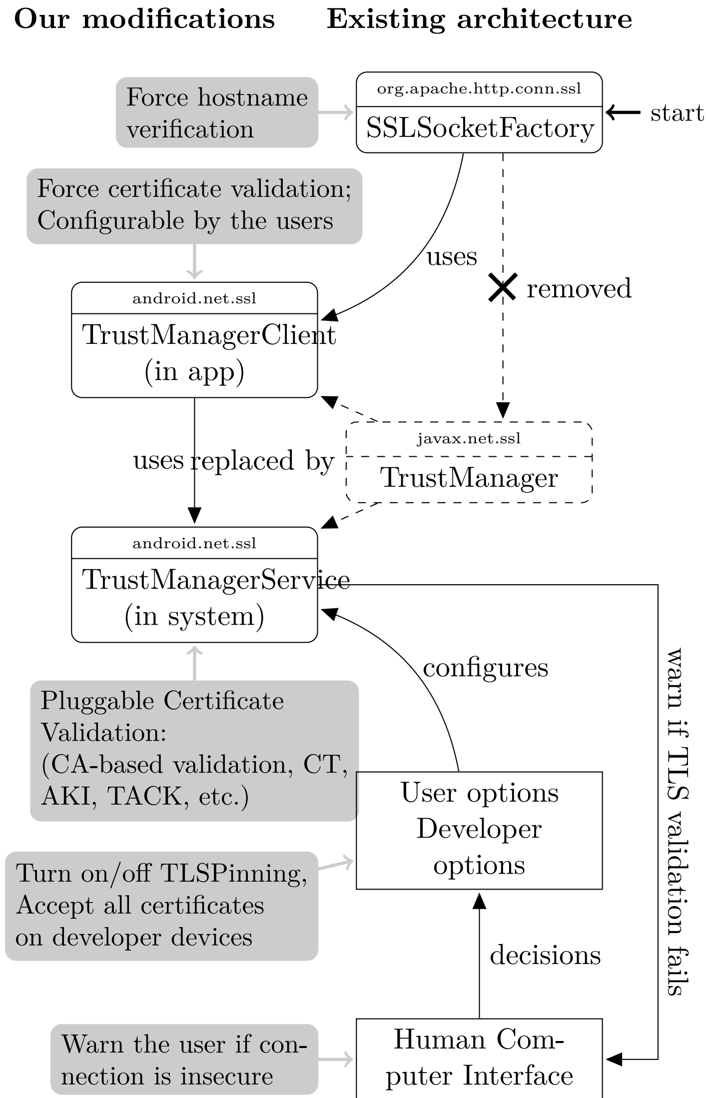
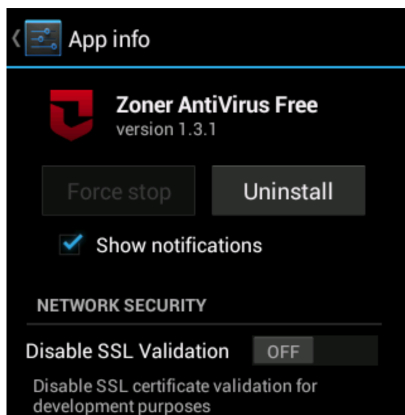


Figure 7.3: This figure illustrates the process of creating an TLS protected network connection. The grey boxes comment on our contributions.

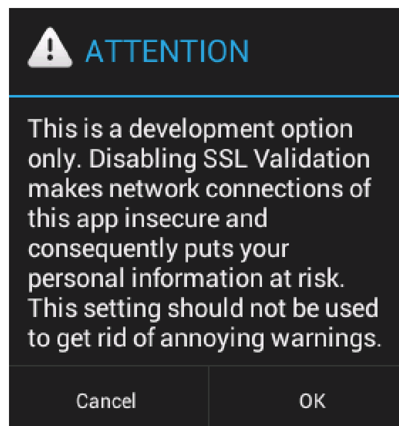
turn off TLS certificate validation for specific apps installed on their device without needing to modify the code of their app. This option is monitored by the `TrustManagerService` and skips certificate validation for this app only. These settings only affect the specific app on the developer device, not the apps deployed onto users' devices or other apps on the developer's device. Thus, even if developers forget to turn on certificate validation again, this has no effect on apps on user devices. This feature effectively protects users from forgetful developers and solves many of the problems we discovered during code analysis and interviews.

We only allow this option on devices that have developer settings enabled. Thus, app developers have a simple way to work with self-signed certificates during devel-

opment while preventing careless users from turning off TLS certificate validation for their apps.[8] Nonetheless, we show a warning message using strong wording that advises against abuse (cf. Fig. 7.4(b)) when this option is toggled.



(a) Developer settings to turn off certificate validation for an app. This option is not displayed for normal users.



(b) On disabling validation, a message warns against security threats.

Figure 7.4: Instead of writing code, TLS parameters can be changed with via Developer settings options.

TLS Pinning

TLS public key pinning can be configured by app developers to limit the number of certificates an app trusts. It can either be used to specify exactly which CA certificates are trusted to sign TLS certificates for this app or directly specify which individual TLS certificates are to be used. The standard Android approach to use certificate pinning requires developers to implement it individually in their source code, resulting in only very few apps implementing pinning at all.[9] The standard approach for limiting the number of trusted issuers is using a custom keystore, which is also complex and requires custom code to load the keystore. Using our extension, it is possible to configure TLS certificate pinning in an Android app's `Manifest.xml` file. This allows developers who know which endpoints their app connects to (this should be most apps) to easily and securely add TLS pinning without having to write any code. Figure 7.5 shows the `Manifest.xml` for an Android app with TLS pinning.

[8]While it is conceivable that users annoyed by warning messages could find information online on how to activate developer options and then turn off certificate validation for a specific app, we believe this risk is fairly low compared to the huge benefit this option brings. Additionally, we recommend limiting this option to devices that are registered with Google developer accounts to prevent normal users from carelessly breaking their apps' TLS security.

[9]One of these apps is the Twitter app by Twitter Inc., for whom Moxie Marlinspike developed the TLS pinning code.

```xml
<!-- SSLConfiguration for my App -->
<ssl>
 <pinning>
  <!-- Only trust my self-signed certificate. -->
  <pin type="leaf" val="18:DA:D1:9E:26:7D..."/>
  <!-- And certificates signed by Verisign. -->
  <pin type="issuer" val="8F:57:5A:C8:5B..."/>
 </pinning>
 <logging level="INFO"/>
 <handle-validation-fails action="Do-Not-Connect"/>
</ssl>
```

Figure 7.5: The new TLS configuration options for an Android app's Manifest.xml file allow developers to easily configure different options for handling TLS. Developers can pin either leaf or issuer certificates, determine if their app should log TLS-relevant information and observe how their apps react to failed TLS certificate validations. By default no pin is set, logging is turned off and apps refuse to connect to hosts for which certificate validation failed.

Pinning individual leaf certificates also allows developers to use self-signed certificates in a production environment in a secure way, which is a requirement several app developers articulated (cf. Section 7.4). In case developers wish to restrict the number of trusted issuers instead of pinning individual leaf certificates, pins for certificate issuers can be added to the `Manifest.xml` file in the same way. The app then accepts only certificates issued by the pinned certificate issuers. To simplify the process of creating certificate pins and adding them to the `Manifest.xml` file, we extended Eclipse's Android Development Tools. This way pins for given certificates can be generated and added to an app's `Manifest.xml` file automatically.

Since TLS public key pinning can be problematic in some rare cases, for instance if a company mandates the use of an TLS proxy[10], we allow for TLS pinning to be disabled globally using the (enterprise) Device Administration API policies. While careless users cannot unintentionally turn off TLS pinning since they do not have access to the Device Administration API, enterprises can configure devices to respect company policies.

Force TLS

Our analysis of the 13,500 most popular Android apps showed that 9,934 (73.6%) used HTTP to contact endpoints that supported HTTPS. The developers thus could easily have upgraded their apps to use TLS by adding a single character to the target URLs (as stated by a developer in Section 7.4 "forgetting" the S after the HTTP could be a common problem across apps). Currently, app users have no way of influencing if an app uses TLS or not. To enable the user to upgrade existing apps to TLS, we implemented an HTTPS-Everywhere [116] like approach for the socket

[10] Some companies use TLS proxy servers to monitor the network traffic of their employees.

creation process on Android. The problem of not using TLS/HTTPS on websites is well-known. To protect the users of desktop browsers, the EFF provides the *"HTTPS-Everywhere"* browser plugin[11] for Mozilla's Firefox and Google's Chrome browser. Based on a set of regular expression rules of known hosts which support HTTPS, HTTPS-Everywhere rewrites requests to these websites from HTTP to HTTPS whenever possible.

We extracted the ruleset database of the current development releases for HTTPS-Everywhere[12] and extracted hostnames from 13,500 Android apps that support either HTTP or HTTPS. The current HTTPS-Everywhere database contains more than 8,900 rewrite rules. In addition, we extracted 4,500 hosts from our set of 13,500 apps that support HTTPS. We combined both databases and are now able to rewrite plain HTTP to HTTPS connections for 11,875 hosts, including the most commonly used hosts on desktop browsers and Android apps. To integrate this ruleset into the Android API, we modified Android's `PlainSocketImpl` class (cf. Figure 7.3): each time a new socket connection is established, the ruleset database lookup is performed and in case a rule for target host and port matches, a plain HTTP connection is replaced with an HTTPS connection. To keep the ruleset database as flexible as possible, we implemented it as a system service, which is running in the background and can be queried by every app. Integrating the ruleset database this way allows us to update and maintain rulesets in a flexible way.

To provide control over TLS enforcement for HTTP(S) to the user, we added new settings options: Android users can now turn HTTPS enforcement on or off (cf. Figure 7.4) for all apps in the global security settings. Additionally, we added a second option to an app's settings to turn on or off HTTPS enforcement on a per-app basis. The per-app setting always overwrites the global setting. If the HTTPS enforcement option is marked, our modified socket implementation tries to replace every plain HTTP with a secure HTTPS connection. In case enforcement is turned off, socket connections are left untouched and the app's implementation is used. If it is not possible to replace an insecure HTTP with a secure HTTPS connection although the user configured TLS enforcement, a warning message pops up saying that a secure connection cannot be established and leaves the user a choice to drop the connection or allow it anyway. For more on warning messages see below.

User Protection

Currently it is entirely up to the developers to implement the UI to interact with the user when something goes wrong with TLS. This has led to a large number of apps silently accepting invalid certificates, crashing or displaying unintelligible warning messages such as: *"Reset your local time to the current time"* when faced with a certificate validation error. The lack of a ready-to-use warning message was also an issue criticized by developers in our study. It should also be impossible for app developers to invisibly accept untrusted certificates without the users' consent.

[11] `https://www.eff.org/https-everywhere` – last access 13.04.2016
[12] `https://www.eff.org/files/https-everywhere-4.0development.2.xpi` – last access 13.04.2016

We offer a system-triggered, standardized warning that gives app users the chance to recognize security threats originating from insecure TLS connections, thus preventing app developers from silently accepting invalid certificates. This capability is needed for apps that connect to endpoints outside of the control of developers and thus might not have trusted certificates, such as mobile browsers, news readers, blog aggregators, etc.[13]. In these cases, the users are allowed to decide if they want to connect anyway after being shown a warning message.

Usability studies on TLS warning messages for browsers [188] show that designing meaningful and effective TLS warning messages is a challenging task. Designing such a system is outside the scope of this work which is why we use Android's stock browser warning message for now. Still, we think that having a standardized warning message that app developers can use to let the user decide what to do with untrusted certificates is a good starting point for such future work. Issues such as habituation need to be taken into account at that point, but showing any warning message is better than allowing apps to silently accept all invalid certificates due to developers' negligence.

While we no longer allow developers to silently accept connections for which validation fails, we do allow developers to be more restrictive and drop connections for which validation fails without allowing the user to override. This is the correct (and default) behavior for most apps where the developer knows which endpoints the app communicates with and these endpoints have valid certificates, such as online banking, social networking and most other single purpose apps. As long as developers exercise due diligence with their server certificates, the only validation errors would be in the presence of a real MITM attack[14]. In these cases, users would be effectively protected from themselves.

The decision of whether a warning message should be displayed or connections should be dropped was added as a configuration parameter to an app's manifest (cf. Fig. 7.5).

Thus, our framework can protect users of multi-purpose apps from developers who would hide warnings and accept untrusted certificates as well as enabling developers to protect their users from accidentally accepting connections from MITM attacks for apps where the endpoints are known in advance.

Alternate TLS Validation Strategies

One feature which offers promising future potential is the capability of our system to plug new validation strategies into the system and thus protect both new and existing apps without requiring a large number of app developers to update their code. This could significantly speed up the adoption of alternatives to the current weakest-link CA based system. We have created a plugin infrastructure for this

[13]One of the developers we interviewed explicitly stated he turned off certificate validation for his app entirely because his customers wanted to connect to blogs with self-signed certificates.

[14]We realize not all developers practice due diligence with their certificates, however we still believe this to be the right default setting. With this setting developers would quickly realize that there is a problem with their certificate and would be forced to update it.

purpose and support Certificate Transparency (CT) [136] and AKI [128] as new approaches to validate certificates.

Evaluation

As Section 7.4 showed, all broken TLS implementations on Android and iOS came about because developers wanted to customize the way their app uses TLS and failed to do so safely. While our new approach to TLS-development in apps closes all the security holes we found, it will only be adopted by developers if all their needs are met and they feel comfortable with configuration instead of coding. To evaluate our approach, we conducted two evaluation studies.

We ran interviews with the 14 developers of our previous developer study to discuss how they perceived our proposed solution and if they would be comfortable using it. Since the number of developers available for interview was fairly small, we also did an extensive code analysis of all custom TLS-handling code in the set of 13,500 Android apps to ensure that all use-cases could be covered by our solutions and thus remove the need for dangerous code-level customization.

Developer Evaluation

We conducted a pre-test study in which novice developers added TLS to their apps and used pinning and self-signed certificates. Since our framework made all these tasks trivial, we decided not to deploy this study on a large scale, since it would not have led to any valuable insights. Instead, we focused on whether the proposed paradigm shift would find the developers' approval and make them feel comfortable with configuring TLS instead of implementing it. We presented our approach to the 14 developers from Section 7.4 and queried them if these features would fulfill their requirements and remove the need to customize the way their app uses TLS. We also asked if they would feel comfortable using configuration instead of coding to add advanced features such as TLS pinning to their apps.

All use-cases of these developers were met with our new design and the reaction of the developers was very positive. They confirmed their previous statements – that TLS development is too complex – and that they very much appreciated anything which would ease the burden. They were particularly positive about the option to use self-signed certificates during development and to use pinning in such an easy way for their production apps. None of the interviewed developers were concerned that they could not fulfill their certificate validation tasks with the provided configuration options. Due to the small number of developers willing to discuss the problems they have with TLS development, we decided to follow up this qualitative study with a quantitative study.

Compatibility Evaluation

To evaluate whether our proposed solution really covers all relevant use-cases, we ran another static code analysis on the set of 13,500 Android apps [76]. We extracted all customized TrustManager implementations and manually analyzed the

semantics of the `checkServerTrusted` methods. Unlike in the previous study, we also analyzed the 2.04% of implementations that customized TLS handling without breaking TLS. While there are not many apps in this category, it is still imperative that these good apps also continue to work as expected with our new approach. In total, we found 3,464 classes that implement customized TrustManagers. We categorized them based on their handling of TLS validation compared to the default procedure. Table 7.2 gives an overview of the customizations we found. We denote a customization that weakens certificate validation with a "−", a customization that strengthens validation with a "+" and customizations that do not affect validation security with a "=".

Customized Implementations	3,464	**Security Impact**
Accept All Certificates	3,098	−
Expiry-Only Check	263	−
Leaf-Cert Pinning	47	+
Add Logging to Default Validation	32	=
Add Hostname Pinning to Default Validation	16	+
Limit Trusted Issuers	8	+

Table 7.2: Distribution of Customized TrustManager Implementations in Android Apps

In the 97.02% of cases where TrustManagers accept all certificates or only check certificates' expiry dates, our approach protects app users from careless developers by enforcing secure certificate validation. As we previously showed in Chapter 6, Page 93, 97.1% of the endpoints in the 1,074 vulnerable apps had valid certificates. In these cases, our modification fixes the apps without any development effort or negative side-effects for the developer or the user. In the remaining 2.9% of cases, the endpoints used by the apps do not have valid certificates. In these cases, our system would prevent the connection, unless the developer installs a valid certificate, updates their application to pin the current certificate or sets the `handle-validation-fails` option in the manifest (cf. Fig. 7.5) to show warning messages. Since all three modifications a developer would have to perform are very easy in our framework, we believe this to be a good trade-off for the broken apps.

While the majority of all implementations turns effective TLS certificate validation off entirely, a small number of developers created beneficial customizations. 0.9% of TrustManagers add logging to the default certificate validation process. While this does not strengthen certificate validation itself, it still is a potentially positive feature that should not be made impossible. We therefore added a configuration option (cf. Fig. 7.5) that allows developers to get log output from the framework's validation process.

0.5% of implementations add hostname verification directly to the certificate validation process. This strengthens an app's security since default TLS certificate

validation does not cover checking the hostname during the TLS handshake outside of an `HTTPSUrlConnection`. This feature is covered in our framework (cf. Section 7.5 above).

In the 1.6% of cases where TrustManagers were used to improve TLS validation (i. e. through pinning a leaf certificate or a CA), the functionality added by the custom code is available as a configurable option in our solution. Thus, we found no custom TLS code which implements a use-case that is not covered by our solution with significantly less effort. We conducted field trials that confirmed this analysis.

Deployability

All our modifications are implemented as part of Android's Java Framework. A system update would thus be the most convenient way to make the new features of our system available to developers and users. All apps built from then on would use this update by default and would benefit from our framework's features. Existing apps' binaries do not need to be modified to benefit. However, it is possible for power users with root access to their device to install our modifications on their devices without having to wait for an official update or having to make major changes to their device, such as flashing the device.

A noteworthy feature of our solution is that it does not break binary compatibility and could in theory be used to protect existing apps as well. Our modifications only affect private final methods of the Java Framework and thus do not break modularity or collide with developer code. This would instantly fix all the vulnerable apps we discovered in our studies. However there are some rare pathological cases which would need to be considered if our system were to be applied to all apps blindly.

Transition Period

With such a high number of users at risk and such a slow/non-existent response time by developers when fixing the vulnerabilities, it might be worth considering activating our framework not only for new apps but for all existing apps as well. This would instantly fix all instances of apps with broken TLS we could find. There are two ways in which our framework could be deployed in such a case. The first approach would simply override custom TLS code. This would work fine for 98% of the 13,500 apps we analyzed and fix all the broken ones. However, the remaining 2% would lose important functionality (such as custom pinning) until the developers update their apps. They would still have standard certificate validation but their custom improvements would be disabled. So while applying our framework for all apps in this way is simple and helps most apps, it would be good to avoid this undesirable side-effect. A second approach could combine our framework's validation code with the app's custom TLS validation code. In this case, the simple rule would be only if both validation methods accept a certificate, then the connection is established. If one of the two validation processes rejects a certificate, a warning

message is shown to the user by our framework.[15] The main issue with this is that users could potentially see two warning messages for the same connection: In those cases where validation legitimately fails and the developers of the app followed best practices and warns the user and the users accepts the invalid certificate, our framework would also warn the user and the user would have to accept again. While this is not a security threat per se, it is a highly undesirable characteristic to have in a system. However, our code analysis has shown that these cases should be very rare when no MITM attacker is present. Thus this could be considered an acceptable trade-off to fix the many vulnerabilities which otherwise will remain unfixed for an unknown amount of time. It might also be possible to get the developers of the "good" apps to update their apps for the greater good by contacting them before the cold-turkey roll-out. Ironically getting the few "good" developers to react is probably easier than getting all the developers with vulnerable apps to react. However, evaluating and discussing the ramifications of this kind of roll-out is beyond the scope of this work and would need to be discussed in the community.

Pathological Cases

There are some pathological cases that need to be considered when activating our framework for existing apps:

IP Addresses Instead of Hostnames The current Android API allows URLs and socket connections to be established with IP addresses instead of hostnames. Using an IP address has the drawback that hostname verification might not work properly. The concept of virtual hosts for HTTP(S) servers hinders effective hostname verification when an IP address is used to establish a connection instead of a hostname, since common names for TLS certificates typically are fully qualified domain names [171]. Yet, there are certificate authorities[16] that issue certificates with IP addresses as the common name. So, while creating a secure connection using an IP address can cause problems because hostname verification fails, there is also a valid use-case for this practice. Our TLS API treats IP addresses as normal hostnames during hostname verification.

We analyzed the set of Android 13,500 apps to find those that include URLs using IP addresses instead of hostnames to estimate the scope of this practice in Android apps. In all 13,500 apps, we found 163 apps (1.21%) that include IP address-based URLs, pointing to 118 different hosts. 88 of these IP addresses did not support TLS. Of the 30 which supported TLS, only one of the remaining used a certificate for which hostname verification did not fail. However, this certificate was self-signed, so none of the IP address-based apps used TLS correctly.

Custom Sockets If an app implements a custom application layer socket that resolves hostnames by itself, it may rely on the SSLSocketFactory to create an TLS-

[15] For the cold-turkey approach, the default warning mechanism would be switched from "drop connection" to "warning" since we cannot automatically tell in advance if the app has a legitimate reason to connect to untrusted hosts.

[16] e. g. `https://www.globalsign.com/` – last access 13.04.2016

secured socket based on only an IP, but cater for hostname verification itself at a later stage. Given the modifications of our framework, this implementation would break, as our modified SSLSocketFactory would attempt to verify the hostname (in this case the IP address) during the handshake and fail because the hostname in the certificate presented by the server is unknown to the SSLSocketFactory. While it is uncommon to not delegate hostname resolution to the operating system, we acknowledge that such implementations would need to be updated to work with our modifications.

7.6 Limitations

The studies presented in this work are limited in several ways. We could only download a comparatively small number of iOS apps to manually investigate MITM attack vulnerabilities due to legal restrictions of Apple's iOS AppStore. In contrast to our study of Android apps that tested all popular Android apps with static code analysis, we randomly selected iOS apps of categories of apps we assumed to handle confidential information such as banking and cloud service apps. However, we found that 27% of the investigated iOS apps were vulnerable and could identify popular iOS frameworks with a huge number of apps. Hence, we think that although our iOS study is not as comprehensive as the Android study, we think that our results provide valuable information for understanding the reasons of TLS misuse on appified platforms.

We contacted 78 developers and asked them to participate in interviews about the identified TLS issues in their apps. Only 14 developers agreed to participate in an interview while all others either did not respond at all or turned down our request with reference to company guidelines forbidding them to talk about confidential information. Despite the relatively small number of participants, we were still able to identify a wide range of causes which lead to serious security issues in the context of app development and TLS. After the developer study, we conducted a large scale code analysis and did not find any indication that there were further issues that would warrant more interviews.

7.7 Summary

In this work, we argue for a new way of handling TLS connections on appified platforms, since previous work discovered severe problems in this area. To discover the root causes of these problems, we conducted a study of 1,009 iOS apps to ascertain whether iOS suffers from the same problems as Android. We also surveyed developer forums and conducted a developer study. Based on our findings, we proposed to rethink how developers interact with TLS: instead of requiring developers to work with TLS on the code level, we designed and implemented a framework that allows them to protect their network connections via configuration. Our solution prevents developers from willfully or accidentally breaking TLS, while at the same time giving them easy access to additional features, such as pinning and the secure use of

self-signed certificates. We evaluated our proposal with existing Android apps and showed that all use-cases we found can be implemented more easily and securely with our approach. The feedback we gathered from developers was also very positive.

This chapter illustrated how system designers can support software developers in a meaningful way by providing easy to use APIs to work with security related programming tasks. An important aspect of this work was the integration of developers into the process of rethinking the current interfaces. While system designers are involved in designing easy to use APIs, their widespread adoption and use remains a joint achievement of both system designers and developers.

In contrast, the next chapter will focus on an easy to use mechanism to verify software integrity and authenticity that addresses system designers only (e. g. the operator of an app market). The involvement of developers is very limited and end users will not even realize the presence of the mechanism until a malicious event is detected. This work will show the power of system designers with respect to deploying easy to use security mechanisms and will demonstrate how end user can be relieved from technical burdens.

8 System Designers: Distributing Software in a Bullet-Proof Way

Disclaimer: *The contents of this chapter were previously published as part of the paper "Hey, NSA: Stay Away from my Market! Future Proofing App Markets against Powerful Attackers" presented at* 21th *ACM Conference on Computer and Communications Security (CCS) in 2014 [74] together with co-authors Sergej Dechand, Felix Fischer, Henning Perl, Matthew Smith and Jaromir Smrcek (alphabetical order). As this work was conducted with my co-authors as a team, this chapter will use the academic "we" to mirror this fact. The idea and initial concept for this work came from myself. Developing the system design for the Application Transparency concept and its integration into the Android ecosystem was joint work with Felix Fischer and Sergej Dechand. I crawled the Google Play market, downloaded approximately 1 million Android apps and analyzed their app signing practices. I analyzed the telemetry data provided by the Zoner anti-virus app. My co-authors contributed in different ways. Implementing the Application Transparency framework was joint work with Felix Fischer. Jaromir Smrcek provided access to telemetry data of users of the Zoner anti-virus app for Android. Together with Sergej Dechand and Henning Perl, we compiled the paper for publication.*
Software described in this chapter is available at `https://zenodo.org/record/51119`.

8.1 Motivation

The process of installing software is a highly security relevant action – and in the face of powerful attackers including nation state adversaries, it is currently a leap of faith that the software being installed has not been tampered with. While in the past physical media offered some assurance as to the provenance of software and made it unlikely that the version of software being installed was tampered with in a targeted attack, the move to digital downloads and app markets offers a very convenient way for powerful adversaries to target specific users with customized malware. In the case of digital downloads from websites, users could in theory verify the software being installed by comparing checksums[1]. However, this procedure is hard to use for the average user and never received widespread adoption. Central software repositories, and more recently mobile app markets, require developers to sign software/apps prior to inclusion into the repositories. This offers more convenience and in many cases a higher level of security for users compared to direct downloads from webpages.

However, there is no automated mechanism for users or developers to verify that an app being installed is actually the original and untampered app released by the developer.

Recent revelations have shown how infrastructure organizations can be pressured or attacked by nation state adversaries who want to gain access to specific targets. In May 2014, it became public that parcel services gave the National Security Agency access to routers that were intended to be shipped overseas so they could install backdoors. The packages were then resealed and shipped containing the backdoors [101]. In September 2011 the DigiNotar Certificate Authority was attacked

[1] Assuming the channel used to obtain the checksum was not compromised as well.

by the Iranian government and issued fake TLS certificates to attack 300,000 Iranian Gmail users [195]. These two publicly known attacks illustrate the power that nation state adversaries can and will assert on infrastructure and service providers which distribute software and hardware to circumvent security measures.

The central app distribution process via markets or repositories is a similarly tempting target. An adversary in control of an app market or repository as well as an adversary who can coerce an app market or repository can easily distribute custom malware to specific users or groups of users. This is a particularly convenient attack vector since most app markets require personal registration and hence make it easy to identify and target specific users. App markets have already been the target of attacks with the aim to distribute malicious apps in the past [51, 157, 100, 208, 207, 202, 39, 204, 97]. Especially in the light of the limited capabilities of mobile anti-virus apps [197, 164], Android's app ecosystem is an attractive candidate for targeted attacks.

A signature verification process vulnerability discovered in 2013 [20, 18] unveiled that maliciously modified apps may be planted on around 75% of all Android devices without users being any the wiser [8]. This attack allows attackers to tamper with app updates in addition to fresh installs. However, there are more subtle ways app markets can be used to open up attack vectors against specific users: On a regular basis, new security vulnerabilities are being discovered, (hopefully) fixed and updates are released.

However, for an app market it is very easy to withhold updates from specific users, thus guaranteeing that they remain vulnerable to known security issues. While most app markets probably have their users' best interest in mind most of the time, attacks against the app markets themselves and the Snowden revelations necessitate the possibility to verify that app markets and software repositories behave correctly and treat all users equally.

In this work, we conduct an extensive analysis and evaluation of the state of current Android app markets in terms of transparency and accountability during app installation and updating. We show that for both end users and developers, the current model of app distribution is severely flawed and makes targeted (as well as non-targeted) attacks unnecessarily easy.

To overcome the resulting lack of transparency and accountability, we present a new framework called Application Transparency (AT). AT protects against "targeted-and-stealthy" app markets by making attacks against specific users or groups of users easily detectable. AT extends the concept of Certificate Transparency (CT) [135]. Unlike CT, AT provides synchronous prevention instead of only retroactive notification. With respect to app installation and updates, AT guarantees that all users see the same app and version as everyone else and that developers can be certain that no tampered versions of their apps are distributed without their knowledge. Our approach can be applied to any repository or app market based software distribution system. We fully evaluated and implemented our solution for Google Play and use this as an example throughout the rest of this work. We chose the Android app market for the deployment of our solution, as it is one of the largest and most vibrant app markets and has been the target of a number of real world attacks in

the past.

8.2 Background

This section provides background information and related work on the app signing and installation process on Android as well as an introduction to Certificate Transparency.

App Signing on Android

Android apps are digitally signed by signing keys held by their developers [99]. Apps' digital signatures are intended to identify application authors and to establish trust between apps and Google Play. Google's recommendation is to use self-signed certificates with a key size of 2048 bits for RSA keys and a validity period of at least 25 years. They do not accept apps signed with a certificate that expires before 2033. In case developers author multiple Android apps, they are encouraged to sign their apps with the same signing key. Furthermore, app updates are expected to be signed with the same key to enable the detection of malicious updates.

App Publishing in Google Play

To publish apps in Google Play, developers have to package an APK file, sign it and upload the APK file to Google Play's developer console. After adding a description and determining in which regions and for which devices the app should be available, the app goes through internal checks before being published. One such check is Google's Bouncer service which analyzes apps for malware [98]. Other checks include the verification of the signing key's expiration date. After uploading the APK file, a popup is shown to the developer promising to make the app available for download in the next couple of hours. Usually apps or updates are available within 60 minutes after upload. Since many mobile app companies offer apps for multiple platforms such as Android, iOS, Blackberry and Windows Mobile, app building and publishing is often outsourced to third party companies that take care of signing and publishing apps (cf. Section 8.3).

Android App Installation and Updates

Whenever a new app is installed on the user's discretion, Android's "Verify Apps" mechanism checks the app with Google's integrated anti-virus feature, verifies the APK file's digital signature with the developer's certificate and keeps the certificate for checking future app updates.

Updates for apps need to be signed with the original private key that was used when the app was built initially. Otherwise updates will be rejected and users are required to uninstall previous versions before being able to install the app's latest version. Thus, in case developers lose their private key and need to sign app updates

with a new key, their users are shown an error message stating the updates cannot be installed.

Certificate Transparency

Certificate Transparency (CT) is a framework proposed by Google that aims to securely and provably log all X.509 signing activities of Certificate Authorities [135]. The goal is to prevent CAs from creating "attack" certificates which without CT could then be used to mount Man-In-The-Middle attacks against TLS connections with a very low risk of being caught. The basic concept of CT is to offer a publicly available tamper-proof append-only log that contains all CA-issued TLS certificates. Immediately after an TLS certificate is added to the log, the log responds with a Signed Certificate Timestamp (SCT) which represents a promise of the log to include the newly added certificate to the log within the log's Maximum Merge Delay (MMD) time. The log provides two different types of cryptographic proofs: (1) Users of the log can obtain Proofs-of-Presence (PoPs) that allow everyone to verify that a given TLS certificate is part of the log and (2) users of the log can obtain Proofs-of-Consistency (PoCos) to verify that a snapshot of the log is a successor of a previous snapshot. CT knows two different types of clients: (1) auditors are third parties that monitor CT logs for correct behavior, e. g. an auditor checks whether PoCos for a log are correct, (2) client software (such as a browser) which does not directly communicate with a CT log server but relies on the SCTs signed by a log. These clients must trust the correct behavior of a log, which is checked by the auditors. Web-browser users are clients of auditors and can check a CT log's correct behavior by asking an auditor for the results of its consistency checks. A new extension of the CT log is built every MMD and includes all existing and new certificates added within the last MMD. The underlying data structure of CT's log is a Merkle Tree (MT), an append-only tree in which every node is labelled with the hash (typically SHA-256 or SHA-512) of the labels of its children and possibly some additional metadata describing the node. MTs enable the efficient verification that certain data is present in the tree. A PoP requires $\log(n)$ hashes, n being the layer count of the MT, and contains one hash of each layer of the tree. Proving that one snapshot of a MT is the successor of another snapshot can be done in logarithmic time and space by providing one hash per layer of the tree. The value at the root of an MT is signed with the private key of the CT log provider and called the Signed Tree Hash (STH).

Limitations

Asynchronous Proof Validation For performance and infrastructural reasons, CT does not conduct synchronous proof validation. Instead, CT [135] proposes that users gossip received SCTs to detect possible attacks as soon as possible. This mechanism does not protect users when establishing an TLS connection, but uncovers malicious logs (i. e. logs that issue SCTs but do not add the certificates to the log after the MMD) and attack certificates with very high probability over time. This however leaves a time window open for successfully attacking CT users.

Revocation While CT provides PoPs, with CT alone it is not possible to check whether a given TLS certificate is still current and has not been revoked. Revocation Transparency (RT) [134] is a CT extension that allows certificate revocation. RT proposes two different methods to store revocation information. The first one is a so called Sparse Merkle Tree (SMT) that is a regular MT in which most leaves are zero. A complete path in an SMT has length 256 and represents the SHA-256 hash of an TLS certificate. The path ends with a 0 or 1 leaf according to whether the certificate is revoked or not. The second proposed structure to store revocation information is a sorted list organized as a binary search tree. However, Ryan [170] shows that both proposed approaches are inefficient since proofs requiring linear space and time require data sizes measured in tens of hundred of gigabytes, which makes them impractical.

Related Work

In addition to Google's Certificate Transparency framework, there are further academic proposals to provide transparency or tamper-proof logs.

Kim et al. [128] propose a system called Accountable Key Infrastructure (AKI) for TLS certificates. In AKI, so called Integrity Log Server Operators are expected to log publicly available X.509 certificates and push them to an Integrity Tree which is a lexicographically ordered hash tree. Since an Integrity Tree is not an append-only data structure, all operations in the tree are digitally signed and validators are expected to monitor the correct operation of the Integrity Log Server's data structures and perform consistencychecks between different versions of the tree. The Sovereign Keys (SK) system [58] proposes to operate timeline servers that act similarly to timestamping servers [105]. SK runs a Trust-On-First-Use model, i. e. the first registration of an X.509 certificate binds the key to the domain name, subsequently preventing duplicate registrations for the same name.

Ryan [170] proposes to enhance CT with an effective revocation mechanism. He introduces a second lexicographically ordered tree called LexTree in addition to the append-only MT-based data structure called ChronTree. In conjunction, both trees can now provide proofs of presence, currency and absence of data in the log. Adding a second tree to the log requires an auditor to not only prove consistency between two different ChronTree versions but also between the Chron- and LexTree. Consistency-Proofs between Chron- and LexTrees are linear in time and space. Ryan proposes to apply the enhanced CT framework to email encryption.

Related work also addresses app installation on Android devices. Barrera et al. [12] analyzed 11,104 Android apps and extracted 4,141 signing keys. They found that 18% of the certificates sign more than one app. They also operate the `https://androidobservatory.org` service that provides meta information such as requested permissions, version code and information about the signing key for 31,368 Android apps as of February 2014 from Google Play and alternative markets. Zhou et al. [209] present DroidRanger, a tool to detect malicious apps in popular Android markets. They propose a permission-based and a heuristic-based scheme to detect malware in Android app markets and come to the conclusion that

the evaluated markets are functional and relatively healthy. Zhou et al. [208] analyzed the occurrence of repackaged Android apps in six popular alternative app markets and found that 5% to 13% apps in those alternative app markets were repackaged. The problem of repackaging of Android apps was also investigated by Zhou et al. [207]. Vidas et al. [197] analyzed repackaging in alternative markets and found that some markets exclusively distribute repackaged versions of apps containing malware. Permission dialogs allow users to decide whether they agree to an app's permission requests. Currently, those dialogs are the only mechanism that protects users against too permission hungry apps. Felt et al [159] found that many developers over-privilege their apps and that users have problems to correctly understand Android's permission dialogs or even do not read them at all [84].

8.3 App Signing Practices

In the following section we evaluate app signing and packaging strategies currently employed in Google Play and other alternative markets. Therefore, we analyze the signing practices of 989,935 distinct free Android apps from Google Play (97% of all free apps in Play as of April 2014)[2].

For these apps, we extracted the signing keys to evaluate Google Play's current app signing practices. We extracted 380,345 distinct certificates that were used to sign all apps in our corpus.

380,285 (99.98%) of the signing keys are self-signed, 5 apps were signed by one single certificate issued by a dedicated Android CA from Symantec. We found three certificates signed by CAs from major telco providers such as Sony Ericsson, Cisco and Samsung. The remaining 47 certificates were signed by developers' custom CAs.

Table 8.0(a) summarizes the algorithm suites employed by the certificates we analyzed. While 369,278 (97%) certificates apply current security best practices and secure algorithm suites, 2860 certificates employ MD5 and 23 certificates use MD2 as their signature algorithm and hence unnecessarily weaken their signature security.

51.6% of the certificates follow Google's security guidelines and have a key size of 2048 bits (cf. Table 8.0(c)). 0.1% of the certificates employ larger keysizes and 48.15% employ 1024 bit keys. However, 277 certificates use 512 bit RSA keys and hence undermine their apps' security and Android's security guidelines.

Google recommends a validity period of 25 years or longer for signing keys and requires that apps published in Google Play must be signed by a certificate with a validity period ending after 2033. Table 8.0(d) illustrates the distribution of validity periods in our app corpus.

While 1% of the certificates have a shorter validity period than recommended, 70% are within the recommended validity period (25 – 50 years). 12.3% of the certificates have a rather optimistic validity period between 100 and 1000 years and 2033 of the certificates are valid for a 1000 years or longer. The longest validity period we found is a certificate that is valid until the year 10,049. We found 9 certificates that expired before 2033; 4 of them issued in 2013. The most recent of

[2]Due to geographical restrictions, we were not able to download the remaining free apps.

(a) Deployed Signing Algorithms

Algorithm	Certificates	Affected Apps
sha1WithRSA	215,004	584,901
sha256WithRSA	154,274	284,607
dsaWithSHA1	8,154	20,600
md5WithRSA	2,860	16,669
sha512WithRSA	28	33
md2WithRSA	23	82

(b) Apps Signed per Certificate

Apps Per Certificate	Certificates
< 5	390,254
5 – 10	10,967
11 – 20	3,756
21 – 50	2,292
51 – 100	673
101 – 1000	463
1001 – 10000	16
> 10000	4

(c) Deployed Key Sizes

Key Size	Certificates	Affected Apps
16384	3	3
8192	35	65
4096	599	1,289
2048	196,305	398,322
1024	183,126	506,477
512	277	743

(d) Validity Periods of Certificates

Validity Period	Affected Certificates
< 25 years	3,839
25 years	101,691
26 – 50 years	164,763
51 – 100 years	61,065
101 – 1000 years	46,954
> 1000 years	2,033

Table 8.1: App Signing Practices

these certificates was issued in August 2013. Hence, Google Play's enforcement of its own guidelines seems to be rather laissez-faire.

Table 8.0(b) illustrates the distribution of signing keys and the number of apps signed by a specific key. 92.28% of all keys are used to sign one unique app. However, we found 483 signing keys in our app corpus that signed up to 25,190 apps. Interestingly, 0.1% of the keys we analyzed signed 113,842 (11.5%) of the apps in our corpus.

Certificate	Apps
L=Seattle/O=Qbiki Networks/OU=iOS/Android Development/CN=Andrew Vasiliu	25,190
C=CA/ST=MB/O=Andromo.com L=Winnipeg/OU=Development/CN=Andromo App	10,803
C=RU/ST=NSO/L=Novosibirsk/O=BestToolbars/OU=Desktopify/CN=Anton	15,269
C=US/ST=California/L=Chico/O=Bizness Apps/OU=Bizness Apps/CN=Andrew Gazdecki	13,439

Table 8.2: Certificates that were used to sign more than 10,000 apps

Pathological Cases Four of the extracted keys signed 64,701 apps in our corpus (cf. Table 8.3). Hence, about 7% of all apps in Google Play are signed by these four widely employed keys. While this alone is suboptimal for app security, the fact that the current Android ecosystem has no mechanism to revoke signing keys and their corresponding apps makes things even worse.

These four signing keys belong to service providers that allow their customers to create mobile apps without much coding effort, like e. g. the Qbiki Networks key, which signs apps for the Seattle Clouds[3] service provider. Their customers can

[3]`http://seattleclouds.com/` – last access 13.04.2016

download templates for apps, modify these templates according to their needs and then let the service provider build and publish their apps to app markets such as Google Play or Apple's App Store. During the build-process, all apps are signed with the same private key. The app publishing procedure creates two serious issues for both app owners and their users: (1) The app-building provider has access to all apps' source codes and (2) the signing and publishing process is in its control. Hence, app-building providers can modify apps and push these modifications to official app markets under the radar of the actual developers and users. Given the install counts provided by Google Play, up to 125 million devices have apps installed that were signed by this single signing key.

Alternative Markets

Many apps are not published exclusively via Goole Play, but can also be found in alternative Markets such as Amazon's Appstore or the F-Droid market. Overall, there are more than 30 "official" alternative markets available[4]. In the following, we evaluate app signing and packaging strategies for 45 popular Android apps[5].

Results In our sample set, we could not find a single app of which the same version was available across all the markets we surveyed. In all cases, the Google Play version was the most current (but not necessarily as new as to be the most current version according to official developer announcements), which underpins the assumption that Google Play is the most important app distributor for most developers. 19 of the 45 apps we analyzed were signed with the same key across all markets offering the app. However, although the same signing keys were used, all apps distributed APK files with different SHA-256 values. In 26 cases, at least two different signing keys were employed. An interesting case is the Amazon market that is supposed to be a benign alternative market and offers 24 of the 45 popular Play apps we tested. 15 of those apps were signed with different keys than their counterparts in Google Play. A popular example is WhatsApp. The officially announced most current version[6] is 2.11.169. Google Play offers version 2.11.152 and we found 6 different "most current" versions of the app in 12 alternative markets.

Discussion

Analyzing the current app signing practices for 97% of Google Play's free Android apps and evaluating the update and packaging strategies employed by popular Android app providers unveils an unnecessary intransparency and inaccountability for app users. The fact that single app signing keys are entrusted with more than 25,000 Android apps constitutes a serious problem with verifying the authenticity of Android apps. While Google intended to create a path of authenticity between app

[4]Cf. `http://www.onepf.org/appstores/` – last access 13.04.2016

[5]We chose the 45 most popular Android apps across Google Play and 33 alternative markets, as listed by `https://play.google.com/store/apps` – last access 13.04.2016

[6]As of February 20th, 2014

developers and their apps by enforcing (self-signed) certificates to sign apps, the current trend towards outsourcing app signing and distribution to a few big players in the market undermines this intention. The fact that these service providers use one single key to sign thousands of apps without an effective revocation mechanism at hand jeopardizes both app developers – since one stolen private key may result in harmful updates for thousands of apps – as well as app users, since one successful attacker could plant malicious apps into hundreds of millions of devices. In addition to the app signing bug uncovered in 2013 that still affects a huge portion of Android devices, the fact that Google accepts signing keys with 512 bits for apps and that such apps are actually deployed to Google Play shows that Google's line of defense is fragmentary.

Additionally, we found that even the most popular apps are signed with different keys for the same version across multiple markets, making it hard for their users to reliably determine the authenticity of apps. Not having a straightforward tool at hand for verifying apps' authenticity becomes even more serious in the light of the fact that some alternative markets turn huge portions of their apps into malware and then act as super-distribution points for malicious Android apps [209]. Hence, users of alternative markets have no chance to differentiate between harmless and harmful apps – even if they compare apps' checksums or certificates.

8.4 Threat Model

In the following, we present and discuss our threat model we call "targeted-and-stealthy". Here we assume that reputable app markets are not interested in distributing malware to all of their customers, but might be interested or coerced to attacking specific target users. Secondly we assume that the app market attacker does not want these attacks to become public knowledge. Currently, users who search, download and install apps from centralized app markets and repositories need to more or less blindly trust these distributors. The power to deploy (targeted) attacks held by software distributors in the app market ecosystem is comparable to Certificate Authorities in the world of TLS certificates [195]. Due to their central and trusted position in the software ecosystem, coerced, hacked or malicious app markets have the ability to easily deploy targeted and stealthy attacks against specific users or groups of users. Since apps are signed with developer keys, mass-distribution of tampered (malicious) apps cannot be considered a stealthy attack. App developers could simply verify the authenticity of their own apps and unveil the rogue market's malicious behavior. However, if tampered apps are only distributed to specific targets, it is easily possible to run "targeted-and-stealthy" attacks, especially in the light of current app signing and packaging strategies employed by developers (cf. Section 8.3).

In our threat model we distinguish between app markets and app market clients installed on users' devices. While app markets maintain repositories, app market clients send install requests to markets and perform the actual installation. In our threat model, app markets as technical and legal entities can be compromised or

coerced and thus are our threat actor. In contrast, app market clients are installed on the users' devices and for this work we consider them to work correctly. The current status quo for app market clients is that the installation of new apps must be triggered by users, but updates to installed apps are installed silently without user interaction. This type of silent (update) install can be triggered by the app market by offering a newer version of an already installed app. Hence, app markets cannot conduct silent installs but may trigger silent updates. In this work we do not list kill switches as a threat since these are an OS feature and not an app market feature and thus are out of scope of this work. In general, malicious code in the OS or the app market client is not considered in this work[7]. In summary, our threat model covers the following attacks app markets can mount with a low risk of discovery:

Tampered/Malicious Apps: App markets can offer tampered (malicious) apps to specific users or groups of users.

Tampered/Malicious Updates: App markets can offer tampered (malicious) updates to specific users or groups of users.

Withheld Apps: App markets can withhold applications from specific users or groups of users, for instance to prevent the installation of security software or to perform censorship.

Withheld Updates: App markets can withhold updates of apps (e. g. security patches) from specific users or groups of users, for instance to keep them vulnerable to security issues discovered in older app versions.

Removed Apps: In case that malware is discovered and actually removed from app markets for the masses, the markets can still offer the malware to specific users or groups of users.

The above attack vectors are serious security issues for app market customers. While no targeted-and-stealthy attacks mount by app markets – either voluntarily or under the pressure of a nation state adversary – have become public knowledge, similar attacks in similar ecosystems have been seen in the past [195, 101]. Thus we suggest preemptive action to prevent such attacks from becoming reality.

8.5 Application Transparency

To counter the lack of transparency in current app market installation processes, we propose a new framework called Application Transparency (AT). While Certificate Transparency focuses on TLS certificates, we propose to transfer the core idea of transparency to the world of software binaries in general and provide a proof-of-concept implementation and evaluation for Android applications. We build on the data structures and algorithms proposed by CT [135] and LT [170] and leverage the

[7] This is a similar assumption to stating that web browsers are considered to work correctly when performing TLS certificate validation.

unique properties of app markets to improve previous work and solve the outstanding deployment issues.

The AT framework addresses the security issues presented in the threat model (cf. Section 8.4). For this, AT provides three different kinds of cryptographic proofs that allows users to verify the authenticity of apps provided by app markets. The Proof-of-Presence (PoP) is a cryptographic proof that provides information about the presence of an app in a market. This proof allows users to make sure that a provided app is publicly available and not a targeted-and-stealthy attack by the app market. Hence, app markets cannot (be coerced to) send targeted (malicious) apps to specific users, without these being easily detectable by checking the public logs. Proofs-of-Presence also cover the prevention of silent installations of targeted-and-stealthy malicious updates. The Proof-of-Currency (PoC) is an extended PoP version and provides cryptographic information that allows users to verify the currentness of an app's version. Hence, app markets cannot (be coerced to) withhold certain apps' updates from specific users. PoCs can be utilized as a revocation mechanism. Whenever an app's version should be removed from the log, a new (can be null) hash value is added to the log. The Proof-of-Absence (PoA) provides cryptographic information that allows users to verify app absence messages presented by app markets. Hence, app markets cannot (be coerced to) withhold certain apps from specific users.

AT involves multiple actors as illustrated in Figure 8.1:

App Developers App developers create Android apps and submit them to markets – either to Google Play or one of the many alternatives. In case the app market(s) support AT, the app release procedure for developers does not change. However, if AT is not supported by the respective market, developers have to take one extra step, namely push their app to an AT log. More information on the two cases can be found in Section 8.5.

App Markets App markets accept apps from developers and distribute them to mobile users. App markets supporting our AT framework will additionally distribute corresponding AT proofs (cf. Section 8.5) to their users.

Mobile Users Mobile users install apps from app markets utilizing app market clients such as the Google Play app on Android. Along with the apps, AT proofs are sent to the users' devices. In case AT proof validation is supported by the integrated app market client, users do not have to take any extra steps. In case AT proof validation is not handled by the integrated app market client, users who want to benefit from AT can use the standalone AT app.

Log Providers Log providers operate a pair of two logs that constitute the AT log: A ChronTree and a FixTree (cf. Section 8.5). There are two interfaces, one for accepting new apps from app markets or developers and a second to provide AT proofs. The AT log is rebuilt every MMD to be able to provide the most

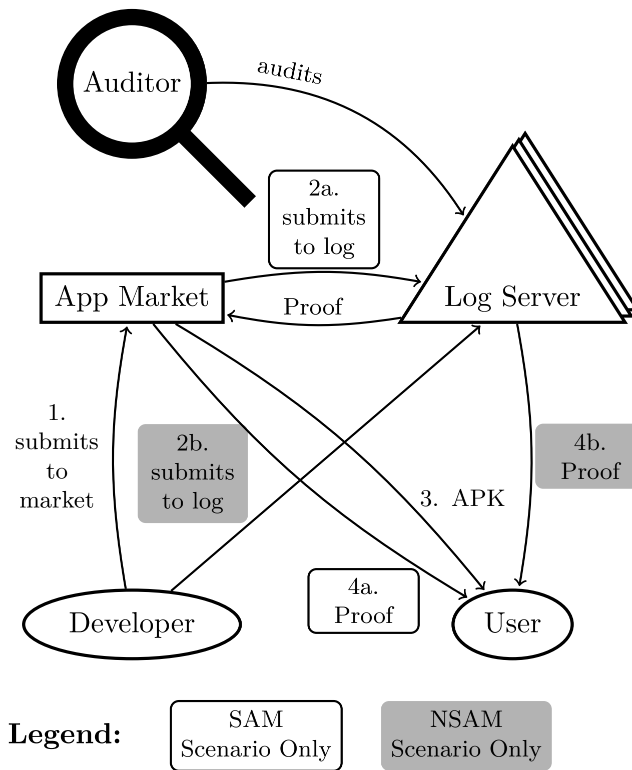


Figure 8.1: All Parties involved in Application Transparency

current proofs. Candidates for log providers are app markets, anti-virus companies or organizations such as the EFF.

By cryptographically verifying Proofs-of-Consistency over time and from different locations, the correct behavior of an AT log provider can be ensured. Hence, users do not have to blindly trust the provider to work correctly – even a nation state adversary could run an AT log without compromising any of AT's security properties. Should a nation state adversary such as the NSA submit an app to their own log, a Proof-of-Currency would be available immediately. Due to the log's append only structure, the app could not be removed again. Even if the NSA revoked the app, it would remain in the log and while a Proof-of-Currency would no longer be available, the corresponding Proof-of-Presence would remain. Hence, the fact that the app had been submitted at one point could not be expunged again.

Auditors AT auditors monitor the correct behavior of log providers and inform the public in case a log provider turns malicious. auditors monitor AT logs by consistently verifying Proofs-of-Consistency. AT log providers cannot be their own auditors, but can audit other AT log providers. Anti-virus companies could act as auditors, as well as for example the EFF. Moreover, app development companies/app generators as well as Android users can act as auditors. Table 8.5 illustrates the different actors and how they benefit from participating in the AT ecosystem.

Actor	Benefits
Mobile User	Targeted Attack Prevention - can be sure that they are treated equally to all other users.
App Developer	Protect their apps against manipulations by app markets.
App Market	Are able to provide cryptographic proofs for their correct behavior.
Log Provider	Provide transparency for the app market ecosystem.
Auditor	Monitor correct behavior of log providers and help to establish a foundation of trust in the AT ecosystem.

Table 8.3: Application Transparency Actors and Their Benefits

Log Structure

The AT log consists of a tuple of two interdependent Merkle trees (ChronTree and FixTree).

As proposed by LT [170], we use the ChronTree which is an append-only and therefore chronologically ordered Merkle Tree. It is extended by appending leaves from left to right, creating a balanced tree. After appending an app to the tree, which is done in constant time, the root hash needs to be re-built. Hence, insertion is $O(\log(n))$. The tree provides Proofs-of-Presence (PoPs) for inserted apps and Proofs-of-Consistency (PoCos) which are able to show that any two versions of the tree are consistent. This is given if one tree is a subset of any later version of that tree. PoC and PoCo checks can be done in $O(\log(n))$. Unfortunately, in a ChronTree PoCs and PoAs demand $O(n)$ since the tree has to show that an app is outdated or revoked.

To make PoCs $O(\log(n))$ and to enable PoAs, we utilize a second lexicographically ordered Merkle Tree [170, 128]. The FixTree[8] is organized as a binary search tree that stores nodes in a way that an in-order traversal yields the certificates stored in the tree in lexicographic order of the certificates' subjects. Insertion, PoC and PoA are $O(\log(n))$ in the average case.

In contrast to the ChronTree, the FixTree alone is not able to provide consistency proofs in $O(\log(n))$ since it does not have the necessary append-only property. Therefore we use a ChronTree/FixTree pair utilizing the advantages of both trees to finally achieve PoPs, PoCs, PoAs and PoCos in $O(\log(n))$. This is done by inserting an app into the FixTree first, and subsequently adding a tuple consisting of the app and the freshly built root hash of the FixTree to the ChronTree. One drawback of using a binary search tree as the underlying data structure of the LexTree is that insertions can unbalance and degrade the tree. We therefore employ a Merkle Tree built up on a 2-3 tree as the underlying data structure. We call this tree FixTree since it guarantees a balanced tree for inserts in $O(\log(n))$ and equally sized proofs of $O(\log(n))$ for all data in the tree. In contrast to the LexTree proposal, the data (i. e. the package information) is contained only in the leaf nodes. Each leaf node for an app has the form $(package_{<ext>}, (h_{v1}, h_{v2}, ...))$ where *package* is the unique packagename (e. g. *com.google.android.gm*) of an app and $< ext >$ identifies a device-,

[8]The FixTree is an extension of the LexTree [170] concept.

language- and region-based app version[9] – e. g. some apps are only available on certain device types, in certain languages and are limited to specific geographic regions such as the U.S.. The values $(h_{v1}, h_{v2}, ...)$ are lists of SHA-256 checksums for the corresponding $package_{<ext>}$ identifier consisting of different chronological versions of an app. Hence, the checksum lists store all app versions that are available for different devices, languages and regions. Whenever an app is outdated – since a newer version is available – an updated hash for the app's $package_{<ext>}$ identifier is appended to the corresponding checksum list. The size of the list of chronological versions of an app the FixTree keeps is bound by a constant N. In other words, the FixTree keeps only $N - 1$ chronological versions of an app. The chronologically ordered list of SHA-256 hashes for an app's $package_{<ext>}$ identifier is utilized as a revocation mechanism. Whenever an app's version should be revoked, a new (maybe null) checksum value for the $package_{<ext>}$ identifier is added to the log. Hence, the revoked version of an app is not totally removed from the log, but marked as not being the most current version any longer. In order to still be able to search efficiently in $O(\log(n))$, the non-leaf nodes hold the (lexicographic) minimum of the leftmost subtree and the maximum of the rightmost subtree.

Both the ChronTree and the FixTree are re-built every MMD, which we propose to be around 30 minutes. An MMD of 30 minutes provides a buffer for capturing the app publication state of the Play Market. Based on the apps we crawled from Google Play, we found that around 12,000 new apps or app updates are published in Google Play every day. Updating the trees every 30 minutes results in 48 updates of the tree every day. Every log update has to append an average of 250 new apps to the current trees. We measured the time our implementation requires to append new apps to the trees. Our tests showed that 48 updates a day are easily feasible from both a computing performance point of view and with respect to the app publishing parameters Google Play offers.

Construction of PoC Each rebuild of the FixTree potentially changes all PoCs for apps in the FixTree. In order to make retrieving PoCs as efficient as possible, the Merkle audit paths for each leaf are built after every merge and kept in a key-value store to make sure the FixTree does not need to be traversed for every request.

Construction of PoA Given a hash h that is not included in the FixTree, a PoA is constructed as follows (cf. Figure 8.2):

1. Search for h in the FixTree to find the closest leaf l smaller than h.
2. Execute an in-order traversal starting from l to the next leaf r to find the closest leaf greater than h (solid path in Figure 8.2).
3. Supply Merkle audit paths for l and r.

Given the Merkle audit paths for l and r, a verifier can then validate that h is absent in the FixTree as follows:

[9] If required, more information can be added to a package name's extension.

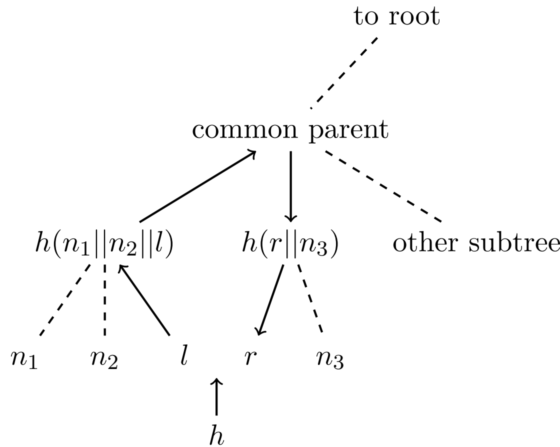


Figure 8.2: Generating a Proof of Absence for h with the adjacent nodes l and r

1. Check that $l < h < r$.
2. For the audit path for l, check that each node in the path is either a right child or the parent is also included in r's audit path.
3. For r's audit path: check that each node is a left child, respectively.

One specific case is constructing the proofs for an element that is smaller or greater than every element in the FixTree. In those cases, the PoA consist of only one audit path of the leftmost or rightmost node respectively.

Deployment

We propose to operate multiple logs – ideally every app market should operate its own log to avoid a single point of failure but still provide transparency for mobile users.

In this section, we show the applicability of AT for Android apps by analyzing various deployment aspects. The deployment is viable without any changes to the app market. However, we also discuss a second deployment option in which the app market participates in the transparency process. We classify these two deployment approaches as (1) the Supportive App Market (SAM) scenario, where the market actively submits its apps to (its own) AT log(s), and (2) the Non-Supportive App Market (NSAM) scenario where apps are submitted to AT log(s) by developers or others. We apply the AT framework to the installation of new apps and to the process of updating installed Android apps.

Adding Apps to the Log

Adding an app to one or multiple AT log providers is essential to benefit from AT's properties. Immediately after receiving an app, the log provider issues a Signed

Application Timestamp (SAT) acting as a promise to add the app to the next snapshot of the AT log and as a proof of the point in time when the app was sent to the log provider. After the MMD passed, all apps that received a SAT in the last MMD interval are included into the next snapshot of the log and all types of proofs for the apps can be requested from the log. The FixTree's leaf node structure (cf. Section 8.5) allows for the inclusion of multiple versions of an app in a single AT log. Chronological updates of app versions can simply be appended to the app version's corresponding list of SHA-256 values. Every (non)-developer is allowed to submit apps to an AT log. While this does not prevent fake submissions, the log's properties make those submissions transparent and public knowledge. Hence, (malicious) fake submissions are publicly detectable and AT's revocation mechanism (cf. Section 8.5) allows for the transparent removal of unwanted apps from the log.

SAM Scenario: In case an app market is supportive and submits all its apps to one or multiple log servers, the inclusion of new apps or updates of existing apps is straightforward: Google Play, for example, runs malware detection and other administrative tasks before a submitted app is made available. Currently, this process usually takes between 60 minutes and two hours. Hence, Google Play could easily submit apps to one or multiple logs without noticeably extending the existing publishing period of an app.

NSAM Scenario: In case an app market is not supportive, there are two alternative mechanisms to make apps available to one or multiple AT log providers. Developers who intend to make their app available in an AT log submit their apps to the log(s) before making them publicly available. To make pushing an app to the AT logs as easy-as-possible for app developers, we propose to include this step into the app building process. Therefore, we extended the `Manifest.xml` file of Android apps [10] and now allow developers to configure one or multiple AT log providers to which the app should be pushed. Based on this configuration, we implemented app pushing into the conventional app building process to make app publishing transparent and easy to use for developers. After packaging the app's APK file with all its compiled code and required resources, the app file's checksum is submitted to the configured AT log(s). The logs then generate a SAT for the app and queue the app for inclusion into the log's next snapshot[11]. The SAT is sent back to the developer. Next, the SAT is inserted into the app's APK file and finally signed with the developer's signing key. This procedure is transparent to developers and does not unnecessarily draw out the conventional app building process. To support developers and to provide a starting point for the Application Transparency system for Android apps, crawling app markets is a pragmatic approach.

Proof-Verification

The verification of cryptographic proofs is a vital part of Application Transparency. In AT, a PoP and PoC or a PoA is a set of two auditpaths that both have the length $\log(n)$ with n being the number of apps in the log. The auditpath length in our

[10] As long as the Manifest.xml file only contains valid XML, Android will not reject apps.
[11] No longer than the log's MMD

current log is 21 and the set of required proofs consist of $2 * 21 * 32 = 1344$ bytes of auditpath data. The average size of an Android app in our sample set is 4,936,800 bytes. This gives an average Proof-To-Payload ratio of $1344/4936800 = 0.027\%$. Proofs are verified during app installations or update requests. Hence, we suggest synchronous security checks. Subsequently, proof-delivery for the SAM and NSAM scenarios are discussed.

SAM Scenario In case app markets support Application Transparency, delivering proofs synchronously is straightforward. On every MMD, app markets need to fetch current proofs for their apps, cache them for the MMD period and deliver them to their users for every app installation or update. Proofs are then sent directly with the APK file. There are no privacy issues or out-of-band connections which could slow down the app installation process.

NSAM Scenario In case app markets do not support Application Transparency, users need to fetch proofs directly from their preferred log(s). As illustrated above, the Proof-To-Payload ratio is rather small. Hence, synchronously fetching the proof does not noticeably slow down the installation/updating process. To speed things up, our implementation triggers proof-fetching as soon the packagename and version code of an app are available, which happens before the download is complete. Additionally, the proof verification process can be split into two more scenarios:

Android OS Integration In case proof verification is integrated into the Android OS, it is straightforward. As soon as the proof is available (either delivered from Google Play or fetched directly from an AT log provider), proof verification can be performed within Android's default app installation/update routine. We implemented proof verification into Android's "Verify Apps" routine that is executed on installation and update. Our implementation verifies a given set of AT proofs for an app and communicates the verification result to the user.

Standalone App In case proof verification is not integrated into the Android OS, a standalone app can perform proof verification. Similarly to conventional anti-virus apps on Android, our app is triggered whenever a new app or an update for an existing app is installed. Our app then fetches the proof for the given app (update), verifies the proof and communicates the verification result to the user. To protect the users' privacy, we added a TOR opt-in feature to our proof-verification app that allows users to hide their identity from AT log providers.

Benefits over CT While CT is used for the transparency of issued certificates, in AT we offer binary protection of the distributed software packages. Due to the different attack models, AT introduces new requirements resulting in the proposed modification of the proof concepts, but also an easier proof deployment. Our proposed system takes advantage of the AT requirements and offers following additional properties in contrast to CT:

Easier Deployment Certificate Transparency relies on the certificate issuer or owner to push a certificate to the CT log. This list encompasses multiple hundred CAs. As of today, only a few CAs have committed to pushing future certificates to the CT log [53]. However, the long term success of Certificate Transparency crucially depends on the participation of all important CAs which comprises a list of more than 100 companies. In AT, app markets can decide whether they wish to provide transparency and do not require the participation of additional app markets. The participation of GooglePlay, providing more than 90% of all installed apps[12], would cover most app installations in the wild. However, even if no app market participated in the AT ecosystem, app developers could start a bottom up approach and register their apps with an AT log to protect their customers. As long as an app is not present in any AT log, a PoA would be spread, while as soon as the app is present in a log, the PoA would be replaced with a PoP, thus giving users transparency on a per app basis. In CT, as long as no PoP is spread for a website, users need to trust the log provider. Hence, CT's security features take full effect only when all TLS certificates are pushed to one or multiple logs, while AT also offers benefits to early adopters.

Synchronous Proof Validation While AT offers synchronous proof verification, the CT project decided against it. Synchronous validations either requires every web-server administrator to always provide the current version of the proof for their website and deliver it in-band during an TLS handshake, or it requires users to fetch the proof out-of-band. Both options have enormous disadvantages that discouraged the CT project from performing synchronous proof verification. App installation, on the other hand, is a relatively rare event and one which takes more time, so the additional overhead of synchronously verifying the proofs does not extend the installation process notably for the users.

Additional Proof Features In addition to the Proof-of-Presence (similar to CT' certificate proof) that enables app markets to cryptographically prove that a presented app has actually been submitted to a log, our system introduces new crucial features to prevent the following attacks:

Withholding Apps Every query to an app market[13] which does not return a positive result and the corresponding PoP has to return a PoA – either provided by the app market (SAM) or the AT log provider (NSAM) – showing that the requested app has indeed never been submitted to that market and the corresponding log. In case of the targeted withholding of an app, a PoA can not be provided, since the app is actually present in the log (and the market) – i. e. a PoP exists – but was withheld only from the attacked user. The non-existence of a PoA and the withholding of the corresponding PoP makes the attack immediately detectable.

[12]The number is based on the meta information we gathered with Zoner's telemetry program.

[13]A query is considered to be the full package name of an app plus its extension value (cf. Section 8.5 for the AT log structure).

Withholding Updates Every update query requires a PoC – either provided by the app market (SAM) or from the AT log provider (NSAM) – guaranteeing that the offered application is the most current version and is also issued to all other app market users.

Evaluation

To evaluate AT, we implemented both the ChronTree and FixTree, a standalone Android app that can verify AT proofs for new apps and updates and an integration into Android's OS-level "Verify Apps" feature. We also deployed the system in the NSAM scenario by integrating AT into the telemetry feature of Zoner's anti-virus app for Android.

Gathered Telemetry Data

Over a period of four months from January 2014 to April 2014, we gathered the following meta information from 253,819 devices that participated in the telemetry program and who gave their consent to anonymously analyze the data for our research:

Pseudonym We assigned a 256-bit random pseudonym to each device to protect the users' privacy. The pseudonym did not reveal any private information.

DeviceInfo We collected manufacturer- and device model information as well as the installed Android version.

DeviceFlags We gathered three different flags for every device: (1) Whether developer options were enabled, (2) whether app installs from untrusted sources were allowed and (3) whether USB debugging was enabled.

PackageInfo For every (pre-)installed app we gathered the package name and version code.

PackageHashes For every (pre-)installed app we gathered SHA-256 checksums of the packages and their corresponding signing keys.

AV-Result For every (pre-)installed app we collected the AV detection result.

Results

The telemetry program gathered the above meta information. Whenever we found only one checksum value for a <packagename, version> tuple across all devices, we treated these findings as harmless, since everybody had the same binary installed and thus no tampered binaries were installed for specific targets. However, when multiple checksums were present across devices, we treated these findings as checksum conflicts. A checksum conflict can have different root causes: (1) App developers compile different versions of an app for different app markets (cf. Section 8.3), (2) apps got repackaged to (benignly) add or remove certain features from the original app or (3) apps are turned into malware to mount (targeted) attacks. In all three cases, users would benefit from AT's transparency features. While most checksum conflicts we found fall into categories (1) and (2), in combination with anti-virus

software AT can help to assign apps to categories (1) and (2) and to distinguish apps that fall into category (3).

App-Type	Amount
System Apps	8,632
Pre-install Vendor Apps	15,999
Third-party Apps (Google Play)	46,481
Third-party Apps (only other Markets)	17,078

Table 8.4: App Checksum Conflicts across all Devices

Signature Keys	Result	Amount
Same Key	`Trojan.AndroidOS.Stealer.A`	2
	No Information	21,623
Various Keys	Permission Remover Service	2
	Adware	4,657
	Rootkits	788
	Negative	5,241
	No Information	4,265

Table 8.5: Google Play Apps with Checksum Conflicts

We collected information for 912,393 different Android apps[14]. While 824,203 apps had no checksum conflicts, we found 88,190 apps (10.7%) with checksum conflicts as shown in Table 8.5. Here we distinguish between three different types of apps: (1) System apps signed with the same key as the Android SDK, (2) vendor apps which were pre-installed by the vendors and (3) third-party apps installed by the user.

Since Google Play is the most important and most widely used app market for Android, we chose this market as our transparency baseline, i. e. we compared the apps' checksums with the values we found in Google Play. Although Google Play could theoretically have provided us with some tampered apps when we crawled the market, for our further analysis we assumed that Google Play played fair with us. Table 8.5 shows the anti-virus results for the conflicts for apps distributed by the Google Play store.

The checksum conflicts result from the usage of different keys for different markets, app customization, or in the open source case, different distributors. Another explanation for conflicting checksums for apps signed by the original signing key would be targeted attacks deployed by the original app developer. Although this is highly unlikely in most cases and we could not find supporting evidence, this circumstance reveals the lack of intransparency of current mobile app distribution: Many of these cases cannot be fully assessed without costly analysis of every suspicious application for malicious behavior on a per app basis.

For the 5,445 apps for which we found conflicting checksums signed by multiple keys, our AV app yielded (partly) positive malware detection results, i. e. for either all versions or only some versions, the AV app tagged apps as either adware or other

[14]Each <packagename, version> tuple was treated as a single entity.

malware. 4,657 of these apps were tagged as adware – for 760 apps the official Google Play store version was tagged as adware and the non-official versions were detected as non-adware. These apps pointed to tools that remove ad libraries from existing apps and recompile the original apps. 3,897 apps were detected as non-adware in the original version, but were found to be adware whenever signed by a different key. Hence, in these cases installing apps from alternative markets or off-site resulted in catching adware on a device instead of installing the original app version. 788 apps were detected as rootkits. In all cases, the official Google Play store app versions were detected as non-malware while off-site installs were malicious. These results confirm previous findings [197]. In these cases AT would have protected the users from accidentally installing malware on their devices by informing them that they were dealing with non-publicly known versions of an app.

Discussion

Analyzing AV telemetry meta-data of 253,819 real world Android devices shed light on the current status of apps' intransparency in the wild, demonstrated the need for an effective tool that allows to verify apps' authenticity and confirmed the smooth deployability of AT. For 89.3% of the apps we analyzed in the wild, we did not find suspicious patterns. Hence, most of the installs in the wild are already transparent although no "everything's-logged" cryptographic proof such as provided by AT is available. These installations directly benefit from an AT deployment and give users and developers the certainty that they were not subject to a targeted-and-stealthy attack by the app market.

However, also the remaining 10.7% installations for which we found conflicting checksums would benefit from widespread AT deployment. A huge portion of apps with conflicting checksums resulted from the signing and packaging strategies employed by many app developers (cf. Section 8.3). In theses cases, having AT at hand would allow users to rely on the authenticity of apps, as well as allowing developers to make sure that users of AT do not accidentally install tampered versions of their apps. A rather small fraction of apps we found in the wild were tagged as malware by our AV app. These cases would benefit from a widespread AT deployment as well: We assume malware apps would not be submitted to publicly available AT logs and hence would not be valid installation candidates for users. If malware were submitted to a log, in contrast to current malware detection, tagging a malicious app as malware once and then (transparently) removing the app from the AT logs by using AT's revocation mechanism (cf. Section 8.5) would immediately protect all users of the AT infrastructure. Finally, we found very few conflicting checksums for which we currently cannot be certain whether our findings are harmless or actually malicious. While this is a limitation of our evaluation, it urgently illustrates the problem of current app deployment: There are cases in the wild that cannot be reliably assessed with current tools. On the one hand these checksums look suspicious, but on the other hand the limited capabilities of current malware detection mechanisms make a final decision whether malicious apps were found or not a gambling game. A widespread AT deployment would effectively disclose attackers that try

to invisibly smuggle malicious apps into markets. Consequently AV providers could focus their malware detection efforts on the apps present in public AT logs which eases the development of new, more effective off-device detection mechanisms.

8.6 Summary

The installation of software is a security critical task and the current centralized app market paradigm present in the appified world is boon and bane together and offers powerful attackers a lot of very convenient ways to plant malicious software on users' devices. We illustrated parallels between app markets and similar ecosystems that have already been exploited by nation state adversaries and revealed the urgency to equip app markets and other software repositories with an effective countermeasure. By analyzing signing and packaging strategies of 97% of the Android apps in Google Play, we illustrate that current signing practices directly threaten mobile security and indirectly make it almost impossible for app users to verify apps' authenticity with current tools. We found evidence that a handful of signing keys, used to sign more than 7% of all apps, are not under control of the actual developers and enable a handful of app distribution providers to stealthily create malicious updates. Additionally, we found that pushing different app versions, signed by different signing keys – which leads to different checksums across multiple markets – is a common practice and makes apps' authenticity verification even more difficult.

We then presented the AT framework: an effective mechanism to protect users and app developers against "targeted-and-stealthy" app market attacks. We show that the central software distribution paradigm makes the deployment of AT easier compared to other transparency approaches such as Certificate Transparency (CT). We discuss two deployment paths and show that AT is easier to deploy than CT even if app markets do not support AT. However, AT also provides better security compared to CT due to synchronous proof validation and the availability of proofs of currency and absence.

In an extensive field study we analyzed app metadata from 253,819 real world Android devices that participated in Zoner' anti-virus telemetry program. We found that 90% of all apps would directly benefit from AT by being able to present "everything's-logged" cryptographic proofs. However, also the remaining less transparent cases would benefit significantly from AT. A huge fraction of the currently conflicting cases could be clarified by applying AT: The harmless conflicting apps would become apparent by submitting them to AT logservers. Notably, we also found cases when AT would have prevented users from (unwittingly) installing malicious apps.

The previous chapters on usable security and privacy for end users, administrators, developers and system designers provided novel and in-depth insights into im-

portant and ongoing challenges in our research. The investigations and user studies involving the different actors of IT ecosystems forcefully illustrate the importance of considering end users, administrators, developers and system designers in usable security and privacy research. The interdependencies between the different actors emphasize that collecting valid results from each class of actor poses challenges for usable security and privacy researchers. While conducting user studies with end users, administrators and developers, I learned that the process of data collection is very different when the desired participants are administrators and developers in contrast to end users. Platforms such as Amazon Mechanical Turk allow for the straightforward rollout of userstudies to end users. However, comparable platforms for userstudies with administrators or developers do not exist. Therefore, recruiting a large number of participants for a userstudy is straightforward for end users but a lot more challenging for administrators and developers. Independently from the class of users we wish to study, coming up with a study design that allows to collect information in an ecologically valid way is challenging for researchers.

Across a broad range of related research, many researchers were concerned with the ecological validity of their study designs. Important open questions in that context for example include in which way self-reporting questions influence the results of a usable security and privacy study, if it is valid to use computer science students as participants in studies that involve programming or software configuration tasks or if collecting security and privacy sensitive information such as passwords produce reliable and applicable results. Open foundational challenges are not limited to the above questions, but received limited attention in the usable security and privacy research community. However, conducting more foundational research that allows for a more reliable application of study results is of crucial importance for our community. Therefore, the last chapter of this dissertation discusses an exemplary user study that focuses on the ecological validity of a common class of end user studies. This study is intended to provide first insights in how to investigate questions of ecological validity for usable security and privacy research.

9 Closing the Ivory Gap: Ecological Validity

Disclaimer: *The contents of this chapter were previously published as part of the paper "On The Ecological Validity of a Password Study" presented at* 9th *Symposium On Usable Privacy and Security (SOUPS) in 2013 [75] together with co-authors Yasemin Acar, Marian Harbach and Matthew Smith. As this work was conducted with my co-authors as a team, this chapter will use the academic "we" to mirror this fact. Matthew Smith and I developed the idea and initial concept for this work. The design of both the online and laboratory study was joint work with Matthew Smith; I conducted both studies myself. Analyzing the real world password corpus we accessed for our research was joint work with Marian Harbach. As described in the paper, analyzing the passwords that were collected during the study was done by Marian Harbach, Matthew Smith and myself. Conducting the statistical analyses was joint work with Yasemin Acar and Marian Harbach before we compiled the paper for publication.*

9.1 Motivation

Passwords are the most common, widespread and possibly the most debated authentication mechanism in use. The inherent conflict of creating usable (e.g. user memorable) but secure passwords has kept security researchers busy ever since the introduction of passwords to computer systems in the 1960s (cf. Section 2.3.1). A lot of password policy and password advice is based on anecdotal evidence and theoretical security measures. However, particularly the last few years have seen an increasing number of academic studies into password security and usability. Password studies can be divided into two major categories: studies of real world passwords (usually based on leaked/stolen password lists such as the RockYou and MySpace password databases) and user studies.

The obvious advantage of the first type of study is that the passwords in question are real and thus any results obtained from the study are based on accurate real-world data. However, these studies of course only shed light on the system the passwords were created in and do not allow researchers to experiment with different settings. As Kelley et al. [124] point out, there is also an ethical conundrum, since these password lists were obtained through criminal activity.

User studies offer the advantage of being directed by the researchers so different conditions can be used to study the effects of certain aspects of the password system, thus giving researchers the flexibility to study different security or usability aspects in a controlled situation. However, one great concern about user studies is the ecological validity of the study, i.e., do the study participants behave the way users would in real life and consequently, to what extent are the study results relevant and transferable to the real world?

Komanduri et al. [130] summarize this problem nicely:

> "It is difficult to demonstrate ecological validity in any password study where participants are aware they are creating a password for a study, rather than for an account they value and expect to access repeatedly over time. Ideally, password studies would be conducted by collecting data on real passwords created

> by real users of a deployed system. However, due to the sensitivity of password data and the difficulty of partitioning real users into experimental conditions [...] it is difficult to collect the data [...] from a deployed system."

To counter potential problems with ecological validity, researchers have developed different opinions on which form of user study offers the best ecological validity for a given research goal. Many researchers opt for online surveys to increase the sample size and diversity of their survey population. MTurk in particular has gained popularity.

> "Using MTurk allows us to study a larger volume of participants in a controlled setting than would otherwise be possible" (cf. Kelley et al. [124]).

Buhrmeister et al. also state that the MTurk population is significantly more diverse than samples used in typical lab-based studies that heavily favor college-student participants [32]. Similarly, Bravo-Lillo et al. conducted a MTurk study for diversity reasons [24].

However, there are also online studies conducted using a more local population such as presented by Just et al. [118]. On the other side, Haque et al. chose a lab study over an online study because of results they obtained during a pretest [107]:

> "We conducted a laboratory experiment with 80 UTA students. Although a larger number of participants could have been drawn from an online study, we preferred a laboratory study because our pilot study (N=12) showed that a laboratory study would produce more consistent responses".

While there have been many user studies on a variety of aspects of password systems taking different measures to improve ecological validity, to the best of our knowledge there has been no study to examine the impact user study setups actually have on the ecological validity of these studies. In this work we present a study evaluating several user studies in combination with real world data from our university. We conducted several user studies with students of our university who we asked to create passwords for services similar to their university services. With their consent, we then compared the study results to their real-world passwords for the same services using a number of different metrics.

Our results show that less than one third of our study participants created passwords that did not mirror their real-world behavior at all. Additionally, more than 25% of participants actually used their real passwords during the study. We also find that the ecological validity of password studies can be improved by filtering participants using self-reported data and make recommendations for studies focusing on specific aspects of password usage.

9.2 Background

Ecological validity has been a concern for a great number of research projects. To the best of our knowledge, this is the first study concerning the ecological validity

of password creation in user studies with the type of the study as the independent variable and with a within-subjects comparison with real world password data. However, ecological validity has been discussed in many password research papers. In the following, we present a brief literature overview of a selection of password and password system user studies with respect to the form of the study and the authors' thoughts on ecological validity. The list is far from complete, however, it gives the reader an overview of the vast spectrum of possible ways of running password user studies. We categorize the studies by the following attributes:

Description

It is believed that the description of a study can influence user behavior from the beginning. Some studies try to disguise their interest in passwords, hoping to not create a bias in their subjects: Haque et al. state:

> "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" [107].

This sentiment is found in many studies. Another example is the work of Shay et al. [176]:

> "Ecological validity in many password studies is limited by the fact that participants are aware they are using passwords for a study, rather than for accounts they value or expect to use long-term".

A common approach is to ask participants to role-play a situation where password creation is just one step among others.

However, some studies openly state their interest in passwords or aspects of a password system [88, 95, 118, 178]:

> "Before beginning the experiment, participants were asked to pretend the passwords they create during the session were going to protect their online bank accounts, and they should create passwords that would be easy to remember but hard for other people to guess" [88].

Another paper added an interesting twist to this issue: Kelley et al. [124] had users set up one password for the study website, i. e., a real password, and subsequently had users role-play the creation of several further higher value passwords.

Study Type

Another aspect where different choices have been made is the type of study used. There are many options: The two most common choices are online and laboratory studies. However, there are also a few pen & paper based studies, as well as field and interview studies. Again, authors have different opinions on why they chose a particular type of study.

Many researchers opt for online studies [2, 124, 130, 176, 193]. Common reasons for this choice are the possibility of increasing the sample size and the diversity of the study population in comparison to laboratory studies conducted with students. However, there are cases where a paper-based survey was chosen instead of an online survey, for instance in the paper by Shay et al.:

> "While collecting and managing the data would have been easier online, we were concerned that more security-savvy users would be reluctant to provide truthful information if they thought we could link their responses to their usernames" [178].

They also reported: *"While pilot testing the survey, we received feedback that our password composition questions made respondents uncomfortable. Pilot testers expressed concern that we were gathering so much specific data about their passwords that we might be able to determine them. We feared that these concerns would prevent users from taking our survey or cause them to answer untruthfully"* [178].

The study by Just et al. used a combination of online and paper survey [118]. Just et al. intended to study the security and usability properties of the security questions that are commonly used when users forget their password. While the major part of the study was conducted online, the answers to the security questions were written on paper, since the authors were worried that having the participants enter the security-critical answers online would prevent them from selecting realistic questions. They state:

> "Our experimental method presents an interesting option for obtaining more realistic authentication information in an ethical way. Though while the use of pen-and-paper aids us in this effort, the same practice introduces some factors that are difficult to control. For example, the self-assessment of memorability places a significant amount of trust in the participant" *[118].*

This problem is very closely related to studies requiring users to divulge realistic passwords.

A large number of studies use a laboratory setup to study password systems and password behavior (cf. [40, 88, 95, 107]). Laboratory studies have a number of well known issues that can potentially lead to ecological validity problems, such as the fact that users are not in their natural environment and are particularly aware that they are being studied. Some researchers have used this source of potential bias to try and err on the conservative side of their evaluation such as Haque et al. [107]:

> "Finally, we note that the presence of an observer may, if anything, motivate users to create stronger passwords than they might otherwise."

However, they could not capture and discuss whether or not this effect actually took place. Furthermore, if this effect does occur, it is not desirable for all types of studies.

Several researchers studying usability aspects of password systems avoid the problem of users potentially choosing unrealistic passwords by specifying the passwords themselves. This is often done in case the password itself is not the main focus of

the study and the effect of password behaviors is limited for the sake of the study. Examples of this kind of study were done by Shay et al.[176] and Ur et al.[193]. However, even these studies must take ecological validity into account, since the usability of password systems is often effected by the strength of the passwords. For instance, Zakaria et al. preassigned passwords to the participants, but tried to match what users would choose:

> "In order to maintain ecological validity of this experiment, the passwords tested must be memorable; otherwise they would be less likely to be chosen in the real world" [206].

Researchers also attempt to analyze their data in a way that allows the detection of problems concerning ecological validity. Komanduri et al. state:

> "Two indicators that participants may have answered honestly are that their self- reported password reuse was higher in the basic survey condition than in the four other conditions, and that the computed entropy of passwords in these four conditions was significantly higher than the entropy of passwords in the basic survey condition. Both findings are consistent with users picking better passwords to protect a hypothetical email account than to protect a real survey account. Despite this, we cannot conclude that our results completely approximate real-world behavior; because the hypothetical scenario was the same across the four conditions, [...]".[130]

Non-Password Studies

Schechter et al. [173] conducted a study on the ecological validity impact of personal risk and security priming in a phishing study. They conducted a between-subjects study with three groups. Two groups were asked to role-play a banking task. One of these was primed to pay attention to security while the other was not. The third group used their own personal data. Schechter et al. discovered that priming had no significant effect on the security behavior between the two role playing groups. However, there was a significant improvement in security behavior between the group using their personal data and the union of the role-playing groups. We found the same lack of effect of priming in our study and can offer additional insights into behavioral differences between using real and study data for the domain of password studies, as well as offering the new view of a within subjects design with ground truth data.

9.3 A Study of Studying Passwords

Preamble

For this project, we were in the fortunate situation of being asked for consultation by the Identity Management (IDM) team of our University's IT Services concerning their password policy system. In the course of this work, we discovered a unique

opportunity: The IDM system stored up to five unique passwords per user using asymmetric cryptography, so it would be possible to decrypt the passwords to do a security analysis.[1] The passwords belonged to five university-wide services, comprising the identity management itself, eMail, Wi-Fi, campus login, and Web-Single Sign On (SSO). Under the mandate to improve the security of our university's password system, we were provided with an anonymized dump of the decrypted passwords to help find policies that would prevent weak passwords without putting undue strain on the users.

However, in addition to this security analysis we were thus in the fortunate position to – in theory – be able to design a study that would mirror the enrollment process at our university and then be able to compare the passwords our study participants created to the passwords that they actually created for their real services. We therefore approached the Privacy Officer with a suggestion for such a study. The study's goal would be to allow us to study the ecological validity of password studies based on this data. It would be prepared and run just like a regular password study in which we would ask the students to role-play the enrollment in an university's IDM system. As with all studies we would require informed consent from the participants at the beginning of the study to cover the study itself. However, the final question in the study would ask for an additional informed consent to allow us to compare the passwords our participants just provided with the passwords from their real accounts. Consenting to this comparison was optional and opt-in. We designed the study in such a way that we would never see the account information belonging to any real or study passwords. The analysis of the real and study passwords would be conducted offline and without any demographic data. Only the results of the password behavior analysis would then be linked to the demographic information collected in the study. All results were to be checked with the Privacy Officer before publication. We discussed this study design and its legal and ethical ramifications in detail with the Privacy Officer and the IDM team. Since the study was based on informed consent and the comparison with the participants' real life passwords was covered by a second, separate and opt-in informed consent agreement, our study protocol was approved.

Study Design

Given the wealth of different questions about ecological validity we could try and answer, we had to pick a manageable number to fit into our study. The main question we wanted to answer was: Do passwords generated by participants asked to role-play a scenario in which they have to create a password for fictitious accounts resemble their real passwords? Or do participants behave so differently because of the study that the results of the study should not be used to make inferences about their real behavior? Based on our literature review the two prevalent forms of user studies for passwords are online and laboratory studies, so we decided to study these

[1]While this is non-standard behaviour, this design choice was well-founded and is implemented securely.

two forms of experiments.[2]

Since the password system of our university has password policies in place that force users to create fairly strong passwords, we were concerned that the effect size might be fairly small since the policies rule out simple passwords. Thus we decided to add only one more independent variable to the mix and examine whether openly mentioning that the study is also about passwords has an effect compared to obfuscating the study's purpose. We selected this variable since many papers chose to invest a fair amount of effort to obfuscate their study, specifically stating a wish to avoid priming the subjects in the hopes of getting more realistic results. However, to the best of our knowledge, there is no evidence to suggest that this is a good approach. In fact, it may even be counterproductive.

Altogether, in addition to the two within-subjects conditions of real vs. study passwords, our study covered four between-subjects conditions in two variables (lab vs. online study; password priming vs. no password priming). In all conditions, we asked students to role-play that they had just enrolled in a new university and needed to register for the different services offered by the university. We used the same type of services as offered by our real university.

In both studies, we applied the same password creation policies that are currently enforced for IT service accounts at our university:

- A password's minimal length is 8 characters; its maximal length is 16 characters.
- Password characters are split into four different groups: Upper and lower case alphabetical characters, special characters ,.:;!?\#\%\$@+-/_><=()[]{}* and digits. Passwords that are shorter than 12 characters must include characters from three of the four described character groups. Passwords that are 12 characters or longer only need to include characters from two of the four described character groups.
- Neither the student's first/last name nor the student's ID number may be part of a password.
- Users must use different passwords for all accounts.

Online Study

We invited 16,500 university students via email to participate in our online study, announcing a two-part online study on the creation of online accounts for university services. Participants were told that each part consisted of a simulated online scenario combined with an online questionnaire, taking between 15 to 20 minutes and 5 to 10 minutes respectively. As incentive, we offered our participants the option to enter a raffle for three 100 Euro Amazon vouchers. The email also stated that the second part of the study would follow two days after the first and that they would be able to enter the raffle only after completion of part two.

[2] We did not use MTurk like many other studies have, since we would have lacked ground truth data to compare the behavior for those participants.

To cover the second independent variable, we varied the introductory text. The invitation email was the same for all conditions, inviting students to participate in a study about online account enrollment at a university. After students clicked on the link to enter the study, two different introductory texts were shown. For the non-priming condition, the text just stated that participants should pretend they were enrolling in a new university and should behave as they would in real life. The word “password” was not used at all. For the priming condition, we mentioned that it was important to keep the passwords for the accounts available. We asked the participants to take exactly the same steps they normally take when creating and managing new passwords. We also asked the participants to act as if the passwords for the fictitious study scenario were real passwords. This is the same information about passwords that was used in Kelley et al.'s work [124].

In both conditions, participants were told to imagine that they just enrolled in a new university and intended to use different IT services. Therefore, accounts for the Identity Management System, Email, Wi-Fi and the Campus login service had to be created. The description for both conditions stated that to complete the second part of the study two days later, it would be necessary to log into those accounts again. We included this condition since it is a common approach for researchers to try to urge participants to use passwords they would be able to remember/keep for a while, as opposed to single-use throw-away strings.

After setting up the accounts for the four services, participants for both conditions were redirected to an online survey. The online survey collected demographic information and information about the participants' Internet usage. We also asked the users how they usually manage their passwords. They were also asked how many different passwords they use for all their online accounts to self-report the quality of the passwords they created in the study they just completed compared to their real passwords and if their password creation behavior in the study was different from their behavior in everyday Internet usage.

After two days, our participants received a personalized email requesting their participation in the previously announced second part of the study. After clicking a link contained in the email, each participant was asked to log into the same four services as before, using the password they had created two days ago. After three tries, participants could choose to continue to the next service without successfully logging in, in order to not unnecessarily frustrate our subjects. The system recorded whether or not participants succeeded and how many tries each participant failed. Finally, participants completed a second questionnaire asking how they had managed the study passwords.

Lab Study

We also invited 740 university students to a lab study from our study mailing list. We excluded them from the invitation to the online study, so they did not receive two invitations to this study. Our goal was to conduct a lab study with roughly 70 participants so we invited 740 students, since we usually have a response rate of 10%. We arranged appointments with 75 students of which 68 actually attended

the lab study. The study was set up the same way as the online study, the only difference being that the students had to complete the password creation for the study in an unfamiliar lab environment, with a lab computer and under the supervision of the experimenter. After a brief welcome speech, the lead experimenter read an introductory description of the study aloud equivalent to the online study's description.

While the first part of the study was conducted in our usability lab, the second part could be completed two days later at home. Again, we sent out personalized invitation links for each participant. The participants were told that they would receive 20 Euros each after they completed both parts of the study. Before the first part started, participants had the chance to ask questions or make comments. They were also told that they could request assistance if they had technical difficulties with the lab computer.

Password Analysis

Analyzing passwords is a hotly debated topic. Since our main interest was not in a particular measure of password strength but in researching user behavior, we decided to begin with a manual scoring of different password metrics/patterns.

Expert Scoring

The goal of our manual scoring was to categorize participants based on how similar the metrics of their study passwords were compared to their real passwords. We decided to use this type of review instead of a more algorithmic approach to be able to accommodate the nuanced differences in user behavior that are difficult to capture using formalized rules. We therefore favored a manual approach to explore this aspect. For instance, using metrics alone, it would have been difficult to catch the different behavior for the following fictitious example passwords: Study: "PwdIDM11.", "PwdMail11.", "PwdWifi11.", "PwdPC11." and Real: 'B0ru$$ia09", "16.Januar", "(australien)", "314159Pi". As can easily be seen, the study passwords follow a clear system while the real passwords don't. However, the bit-strength of the password (as calculated according to an approximation of Shannon[3] and NIST[4], respectively) and the crackability (as calculated according to John) is fairly similar. While it would of course be possible to create custom metrics to try and factor in similarity between the password groups, the options would have been endless and unverified. We would also not have been able to capture behavioral anomalies encoded in the passwords such as these real examples from one participant in the study: "studiesSuck123" and "IamSoBored!!!". Since the participant's real passwords did not use such references, this is a case where the attitude in the study differs from the behavior in real life.

[3]cf. Mathematics of Information and Coding, Chapter 2

[4]cf. `http://csrc.nist.gov/publications/nistpubs/800-63/SP800-63V1_0_2.pdf` – last access 13.04.2016

To capture password behavior, we define the following metrics and guidelines aiming to capture a general idea of user behavior instead of pure password strength in an expert scoring process. We break down a password into the following components:

Names Any kind of name, i.e. persons, nicknames, pets, places, etc.

Dictionary Word Any word contained in a dictionary.

Dates Any kinds of date, no matter the form or length, e.g. `1999, 02/03/13, 78, Feb.2., 09081978`.[5]

Simple Numbers Single numbers, counters such as `1,2,3,4` or simple sequences such as `123,456` or `111,222` etc.

Complex Numbers Any combination of numbers that are not dates or simple numbers.

Lower Case String String containing only lower case characters.

Upper Case String String containing only upper case characters.

Mixed Case String String containing mixed case characters.

Special Characters Any combination of special characters.

L33T Speak The use of leet speak.

Keyboard Pattern A combination of characters arising from using adjoining keys on common keyboards, e.g. `QWERTY, sdcx, 7895123` etc.

Random String A random string containing letters, numbers and special characters that could not be sensibly broken down into the categories above, e.g. a string generated by a password generator. We might have misinterpreted strings as random although they followed a structure such as the first letters of words in a sentence.

We considered the following transformation rules to judge the similarity of one password to another:

Ordering and Reordering of Components The order in which components are used

Exchange of Content Any exchange of an instance of a component by different instance of the same component.

Incrementation Any form of systematic incrementation, e.g. `3,4,5` or `!, !!, !!!`

Changing of Case Any changes between upper and lower case for a component or parts of components.

[5]In some cases it was hard to differentiate between numbers and the year in the yy form. In these cases we used the context of other passwords of that participant to try and score correctly.

Insertion An insertion of one component into another, e.g. `password` and `1234` to `pa1ss2wo3rd4` or `password` and `...` to `pa.ss.wo.rd`.

Using these guidelines, each of the researchers scored all participants that had given us permission to compare their study passwords with their real passwords. To assist the manual scoring, we preprocessed the passwords and appended a " [kb]" to passwords which contained a keyboard pattern.

Each subject's password set was assigned to one of the following categories:

Null No apparent similarity between the real passwords and the study passwords

Single There is one study password that is similar to a real password, however, the sets are not similar to each other

Full There are several study passwords that are very similar to real passwords and there is a similarity between the sets as well

System There is a system within each set and the systems are similar, but the composition of the passwords between the sets is not the same

Derogatory Obvious and derogatory reference to the study indicating that the participant did not show normal behavior

The difference between categories Full and System is fairly small. One additional criterion for category System is: If shown eight passwords in random order, is it possible to distinguish two sets of four passwords? If all eight passwords are so similar that it is impossible to distinguish between the sets, the subject is scored as category Full. This scoring system is not designed to measure password strength but likeness/provenance. To put it differently: How useful and accurate are the passwords given in the experiment to study the real life behavior of our participants? This scoring (on its own) does not take strength into account, i.e., rose123 would match Elisabeth9876. We combine this metric with password strength at a later stage.

Table 9.3 shows some examples of our scoring system. The passwords shown there are inspired by real cases, but have been altered so as to not endanger any real user accounts. We discuss each example in the following:

1. In this example, the passwords explicitly reference the study. Since the real passwords are not similar we score this as Derogatory.

2. The real passwords contain a single letter plus a date and a long upper case string with some numbers. Three of the study passwords contain several dictionary words. There is no similarity of pattern at all, so we score this as Null.

3. Two of the real passwords are dates and two are manglings of the same name. The study passwords are dictionary words with numbers and some special characters thrown in, so this is also scored as Null.

#	IDM	Mail	Wi-Fi	Campus PC	Score
1	notsecure12 TreePeter$1	ihatesurveys77 woJIJIui	2moreforyou TreePeter$2	Iknowwhatthisis4	Derogatory
2	EifkLegs KDOSKDO2EWKFD2	CornFlakes	YeaYeaYo U03.03.12	Mineralwater	Null
3	Saver3451 9thFeb90	Lions.Den54 Feb9th90	Plants1.go Peteeer1	Soon,me.1 Peteeeer2	Null
4	Roses220 Mary0908	Roses221	Roses222 Physics2010	Roses223 Maths2010	Single
5	Intovgaad! Intovgaad!	Sydney12 Intovgaad!!	Spain13 !Intovgaad	Hello123 Intovgaad?	Single
6	Fryingpan123 Fryingpan123	Fryingpan456	Fryingpan789 Fryingpan456	Fryingpan99999 Fryingpan789	Full
7	9;6BU7MG3h#y #M24kJB	d<8k@L343oju 38333DI(*DL33T	s$jW7Q639C)H B[L72:7L7cvA	KcL4.,8b7T4A	Full
8	Unlockthis1122655 Secret99499	Unlockthis2233766	Unlockthis3344877 Secret994	Unlockthis4455988 Secret99499!	System
9	Jumpman35 3kefdUed	5JumpmanThree Three5fun	FiveJumpman3 three5Fun	5Jumpman	System

Table 9.1: Examples of the expert scoring process for passwords. The first line in each row are passwords provided in the study while the second shows corresponding real passwords.

4. Both sets of passwords are based on a name plus a number. However the real set is based on name plus date and the name is varied between the passwords. In the study passwords, the name is the same and instead of a date a simple number is used that is incremented over the password. The study passwords are more homogeneous than the real passwords. Thus a singular password is similar but the overall behavior between study and real life is visible and thus this set is scored as Single.

5. This set is the reverse of the above. The study passwords are more heterogeneous but there are singular passwords similar to singular passwords in the real set.

6. In these sets all passwords are very similar. They all use a base word and a sequence of numbers. Thus the set is scored as Full.

7. The same goes for this set. All passwords are of the same nature, i.e. a random string, so this the set is scored as Full.

8. These two sets are generated using the same principle. Both are based on a word plus a number. There are slight differences though. The real passwords are based around variations of the number 99499 while the study passwords use a number pattern with an increment. Thus while no two passwords are the same, it is plausible that the same user created them and it would easily be possible to sort the four correct passwords into the two sets. Thus this set is scored as System.

9. The system in these passwords is also clearly visible. The participant uses the numbers 3 and 5 and a base word and alters spelling and order between passwords. While the similarity is not as high as in the last example it still seems plausible behavior for the user and thus is scored as System.

Scoring Conflicts

Each of the researchers scored the entire dataset separately and in a different order. The scores of all three raters agreed entirely in 47.2% and disagreed entirely in 9.3% of cases. The remaining 43.5% of conflicts had two scores agreeing and could have been solved using majority votes. However, we decided to discuss each participant we did not fully agree on individually. These discussions were conflict free and were usually resolved by the majority explaining the pattern or lack thereof. The final count of the categories is presented in Section 9.4.

Interpretation of Scores

The usefulness of participants for a password study will depend on the research focus of the actual study. If password behavior needs to be studied over several services or passwords, participants in categories Full and System are useful. Our feeling was that participants in categories Full and System both behaved realistically with participants from category Full having more similar passwords in general. While category Single participants can still add value, they can also introduce unrealistic behavior: For instance, they show heterogeneous behavior in the study but have homogeneous passwords in real life or vice versa.

If only a single password is to be studied, our feeling is that participants from category Single are probably acceptable to study. However, it should be noted that the matching password was not always the first participants entered. There were cases where it seemed that the participants used up throw-away passwords until they ran out and then used a real password. However we could not measure this in any meaningful way and thus this feeling should be taken with due caution. Participants in categories Null and Derogatory did not behave consistently and could skew the results of a study in a damaging way.

Apart from our manual scoring, we also applied some traditional password metrics for further analysis and to support our scoring.

Password Composition

Above, we analyzed the structure of the passwords manually and with a fairly coarse granularity. Another measure of similarity for passwords is their composition with respect to single characters. Therefore, we calculated the following composition metrics for every password from both the real accounts and the online and lab study accounts:

- The length of a password.
- The number of upper case characters.

- The number of lower case character.
- The number of digits in the password.
- The number of special characters as defined in the deployed password policies (cf. Section 9.3).
- An approximation of the Shannon entropy for the password.
- The NIST entropy for the password.

In addition to the above metrics we also analyzed our password corpus for the same patterns as described in Section 9.3 algorithmically. For the dictionary check, we compiled a dictionary based on multiple wordlists. These wordlists include Burnett's top 10,000 passwords,[6] lists of first and surnames taken from Wiktionary,[7] an English and a German dictionary, the top 10,000 German words,[8] a list of 85 common emoticons and the following list of study specific words we compiled based on service names and other prominent words. Our algorithm then checked if a password or parts of a password could be matched against the dictionary. Additionally, our algorithm analyzed passwords for the occurrence of leet speak. Leet characters were translated into non-leet speak, then we checked if the translated version could be found in the dictionary. Example: `W@llc0l02` is first translated to `Wallcolor` and then both `wall` and `color` could successfully be matched against the dictionary.

Password Strength

Password strength has been measured in many different ways: From simple 0 entropy, to more elaborate bit strength metrics, guessability and resistance against cracking attacks [21, 124]. There is a fair amount of discussion going on about which metric gives the most realistic measure of password strength for a given type of attacker. In this study, the password strength aspect plays a secondary role since we are mainly interested in the relative comparison between the sets of passwords generated by the same user. We therefore chose the following measures:

Entropy

To compare the relative strength of participants' real and study passwords, we chose two well-known entropy measures. We used an approximation of plain Shannon entropy, i. e., $H = log_2 N^L$ where N is the number of symbols in the alphabet the password is based on and L is the password's length. This approximation of plain Shannon entropy has been repeatedly criticized [21, 124] to not accurately represent a password's strength against an attacker. However, in our case, we were interested in comparing the relative information content of several passwords created by the

[6] cf. `http://xato.net/pass-words/more-top-worst-pass-words` – last access 13.04.2016
[7] cf. `http://en.wiktionary.org/wiki/Appendix:Names` – last access 13.04.2016
[8] cf. `http://wortschatz.uni-leipzig.de/Papers/top10000de.txt` – last access 13.04.2016

same user. To this end, the approximation of Shannon's entropy represents an upper bound of the potential information content of passwords. Furthermore, we also applied the NIST entropy [199] for passwords to get a more conservative estimate of a password's information content. The NIST entropy estimate limits the influence of password length and the use of different character classes while providing an easy to compute set of rules. In both cases, we do not suggest that these measures represent a good measure of absolute strength of a password. We merely wish to compare the values between the study and real datasets.

Crackability

We also compared password strength by subjecting each set of passwords to dictionary attacks using the well-known password cracker "John The Ripper". For all sets, we used three dictionaries: the dic-0947 dictionary that has shown good password cracking performance in related work [199, 198], a list of 220,000 German words from LibreOffice's spell checker, and the over 14 million stolen passwords from the RockYou set which has also been often used [21, 124, 193, 199]. In a second run, we also used the study passwords as a wordlist against the participants' real passwords. Each wordlist was additionally mangled using 1,080 rules from John's "*Single*" ruleset [199]. For the subsequent analyses, we compared how many passwords per subject were crackable using these attacks.

9.4 Results

Participants

Overall, 765 participants participated in our online study and 68 in our lab study. The first 500 respondents in the online study and the first 35 in the lab study were assigned to the priming conditions. Altogether, 75.7% (579) of all online participants and 95.6% (65) of all lab participants completed part two of the study.

We removed the following participants from our evaluation: 85 online and 3 lab participants who did not give their consent that we may compare their real passwords with their study passwords, 8 online and 2 lab participants who did not supply a valid student ID and thus we could not obtain their real passwords and 53 online and 1 lab participant(s) that had only one real password with the IT services department to base our scores on. Since some participants matched criteria in multiple exclusion categories, this left us with a total of 645 records (583 online and 63 lab). Of the 583 participants in the online study, 66% were exposed to the priming condition and of the 63 participants of the lab study 53% were exposed to the priming condition.

Across all conditions, participants were aged between 17 and 55 (23.72 years on average, $sd = 4.31$, median=23), 35.8% were female, 16.3% studied an IT-related subject. Participants self-reported medium IT expertise (average score 3.42, $sd = 1.0$, median=3 on a five point scale anchored at 1=high IT expertise and 5=low IT expertise). The majority of respondents stated that they use the Internet repeatedly throughout the day (90.7%). They reported an average of 18.1 online accounts

(sd = 21.0, median=14). 17.4% had account credentials abused at least once before, only 42 (6.5%) had never forgotten a password before. The majority (79.6%) had forgotten a password at least twice. 63.2% respondents used between 2 and 5 passwords for most of their online accounts and 14.9% used different passwords for all accounts. Participants' passwords in the university IT services database had an average age of 534 days (sd = 391.7 days, median=481). 26.5% used at least one of their real passwords in our study.

Due to a technical problem in condition assignment, participants were not assigned to conditions in a round robin process but sequentially. This had two undesirable effects: first, the non-priming condition in the online study had fewer participants than the priming condition and, second, the average age of the real passwords is lower in the lab study than in the online study (551.2 days online vs. 370.4 days for the lab, medians: 502 online vs 246 for the lab; Kolmogorov-Smirnov-Test for equality, one-tail, alternative=less: $p = 0.0001817$). We tested if removing older passwords would have an effect on any of our tests, but did not find a significant difference. We did not find any demographical differences across our four between-subjects conditions. While the smaller N for the prime-online condition may diminish the sensitivity of our statistical tests, the overall number of participants in the online conditions is large enough to compensate for this. We did not find any significant effects of password age on the password metrics introduced above and could not find any other indication that this confound effected our results.

Scoring Evaluation

The first step in our evaluation was to check whether our categorization described the relationship between the real and the study passwords correctly. We had several hypotheses concerning the correlations we would find in the different categories: category Full participants would have the highest correlation of password composition values between their two password sets of all categories. We expected a weaker correlation for category Single and category System participants and no correlation for category Null and Derogatory participants.

To evaluate our scorings and the hypotheses above, we conducted Kendall's Rank Correlation Tests for all password composition values presented in Section 9.3, the entropy measures introduced in the previous section between the study and real password set as well as the crackability of the passwords. As expected we found highly significant and strong correlations for participants in score category Full and mostly significant correlations in categories Single and Systemas can be seen in Table 9.4. However, it needs to be noted that while we found significant correlations for those three categories, we found no correlation when the entire set of study passwords was analyzed as a whole.

We found no correlation for the categories Null and Derogatory.

To simplify the further evaluation, we conducted tests to see whether we can legitimately speak of Single, Full and System participants, regardless of the condition (online or lab, priming or non-priming) they were in. For this we conducted 2-tailed Kolmogorov-Smirnov tests which are documented in Tables D.2 and D.2 in

	Derogatory		Null		Single		Full		System	
	τ	p	τ	p	τ	p	τ	p	τ	p
Length	.5352	.0464	$-.0439$	.3994	.2157	.0008	.5141	$<$	.0581	.6492
Shannon approx.	.6111	.0247	$-.0368$	.4609	.2006	.0012	.4768	$<$	.0038	.9753
NIST	.1538	.5854	.0778	.1311	.0022	.9731	.2884	$<$	$-.1413$	.2564
Digits	$-.1492$	.5923	.0762	.1523	.3686	$<$	.6528	$<$	.2577	.0541
Upper Chars	.4620	.1030	.1830	.0009	.2584	$<$	.5779	$<$	.1451	.2908
Lower Chars	.1714	.5272	$-.0133$	.7954	.3100	$<$	.6095	$<$	.1200	.3413
Special Chars	.6365	.0301	.3853	.0005	.5376	$<$	.6482	$<$	.3733	.0095
Crackability	.7324	.0250	.1066	.1126	.3352	$<$	.5514	$<$	.0755	.6465

We conducted a correlation test within the categories, comparing study password sets with the respective real password sets. We applied the Bonferroni correction that gave us an alpha value of 0.0063. As expected, we found highly significant correlations in category Full some significant correlations in categories Single and System and rather random correlation behavior in categories Derogatory and Null. This strongly supports our scoring procedure, while also pointing to the limits of assuming the correlation of the above metrics to be very strong between studies and real passwords.

Table 9.2: Password Metrics Real vs. Study (Kendall's τ).

the Appendix D, Page 202. The results show that there was no difference between those conditions with respect to our categorization and thus it is possible to compare the differences in password behavior solely on the category irrespective of the condition. This shows that our scoring was consistent: Participants classified to behave consistently between real and study passwords by our scoring system did compose their passwords consistently, while those deemed to behave inconsistently according to our classification indeed produced independent sets of passwords. This leads us to assume that category Single, Full and System participants behave more realistically in our study than category Null and Derogatory participants, with category Full participants showing the strongest correlation. 26.5% of our participants even used at least one of their real passwords in the study. In the following we refer to the combination of categories Single, Full and System as helpful passwords and the combination of categories Derogatory and Null as unhelpful passwords - in the sense of helpful or not helpful to study realistic user behaviour.

Evaluation

Across all conditions, we found that we had scored most password sets - 46.2% (298) - into category Full i.e., as being very useful for studying password behavior. We assigned 18.8% (121) password sets to categories Single and 5.1% (33) to category System respectively, both in our opinion still representing partially valuable password samples. 28.5% (184) password sets were assigned to categories Null and Derogatory (1.4%), respectively, i.e., passwords that showed abnormal and derogatory behavior. In the following, we will compare how the different conditions affect the results based on this categorization.

Online vs Lab Study

Separating our scoring results by the type of study reveals a trend towards more realistic results in our lab study: More participants fell into the helpful categories Single, Full and System compared to our online study (cf. Table 9.4), the trend

being significant according to Fisher's Exact Test ($p = 0.0296$ cf. Table D.9). These results add weight to Haque et al.'s 12 participants pilot-study's observation that a laboratory study would produce more consistent responses [107]. While these results are statistically significant for our study, this should not be generalized without care. Please check the limitations discussed in Section 9.5 for more information on this.

Score	Total		Online		Lab		Priming		Non-Priming	
Derogatory	9	(1.4%)	9	(1.5%)	0	(0%)	4	(0.9%)	5	(2.5%)
Null	184	(28.5%)	172	(29.5%)	12	(17.9%)	118	(28.0%)	66	(29.4%)
Single	121	(18.8%)	108	(18.7%)	13	(20.6%)	80	(19.0%)	41	(18.4%)
Full	298	(46.2%)	267	(45.8%)	31	(49.2%)	199	(47.1%)	99	(44.3%)
System	33	(5.1%)	26	(4.5%)	7	(11.1%)	21	(5.0%)	12	(5.4%)

Table 9.3: Scoring Results Online vs. Lab, Priming vs. Non-Priming

Priming

Separating our scoring results by the priming and non-priming condition did not show a meaningful difference (c.f. Table 9.4). We verified this by performing Fisher's Exact Test on the 122 primed vs 71 non-primed unhelpful password sets and the 300 primed vs 152 non-primed helpful passwords sets. The null hypothesis that there was no difference in behavior could not be rejected with $p = 0.4698$ (alternative=two-tailed).

Self-Reported Values

We went on to evaluate which self-reported metrics of participants may serve to predict inconsistent study behavior. First of all, we directly asked participants if they behaved differently during the study. Participants that reported different behavior showed significantly fewer counts in categories Full, Single and System and higher counts in category Null and Derogatory as seen in Table D.5. Whether or not a participant failed to remember their password after two days did not have a significant impact on the scores distribution as seen in Table D.8 and neither did participating in the second part of the study as seen in Table D.6.

Overall, participants who changed their usual behavior for the study obtained significantly fewer ratings in categories Full, System and Single and more in Null and Derogatory than participants who did not self-report this, as can be seen in Table D.5. Finally, participants who said that they use individual passwords for each account also scored significantly more frequently in categories Null and Derogatory when participating online (cf. Table D.7).

We also manually analyzed the reasons participants gave for deviating from their normal behavior. We found the following categories:

Disclosure Participants stated that they did not trust us or did not trust others with their real passwords in general.

Memorability Participants stated that they chose simpler passwords because otherwise they would have problems remembering them.

Value Participants stated that they chose simpler passwords because the passwords were unimportant. There was often a reference to it being “only a study”.

Overburdened Participants stated they were overburdened by having to choose four passwords in short succession.

Policy Participants stated that they chose stronger passwords than normal because the password policy forced them to.

Lazy Participants stated that they were too lazy to choose proper passwords, or that they just wanted to get through the survey as quickly as possible.

New Behavior Participants stated that they adopted a new way of creating passwords in general and thus their old passwords were different.

None of the specific reasons for changing password behavior listed above had a significant influence on the participants' categorization as compared to the total of participants who admitted to having changed their behavior for the study.

Consenters vs. Non-Consenters

Altogether, 88.6% of all online participants and 95.6% of all lab participants gave their consent to compare their real passwords with the study passwords. We analyzed if participants who did not consent to the comparison with their real passwords showed any demographic deviations from the ones who did consent. We only found that those participants reported to have different passwords strategies: They stated that they use individual passwords per account more frequently, as shown in Table D.7. We performed two-tailed Kolmogorov-Smirnov tests to see if study passwords

	P-Value
Length	$p = 0.6183$
Shannon approx.	$p = 0.5852$
NIST	$p = 0.9408$
Digits	$p = 0.6352$
Upper Chars	$p = 0.0648$
Lower Chars	$p = 0.3119$
Special Chars	$p = 0.9803$
Crackability	$p = 0.9895$

Table 9.4: (Study) Password metrics for Consenters vs. Non-Consenters (2-tailed Kolmogorov-Smirnov).

supplied by participants who consented to our comparison with their real passwords have similar metrics as the study passwords of non-consenters. The above p-values suggest that there are no statistically significant differences between the two samples for the measured metrics.

Participants vs. Non-Participants

Due to the nature of our password ground truth data, we can also estimate how well our study participants represent the entire population of students to a certain extent. Since our university's IT services provided us with an anonymized set of passwords for all students enrolled for IT service accounts. We calculated average password length, entropy measures, the number of upper, lower and special chars and digits for this set and the set of students that participated in our study. We then conducted 2-tailed Kolmogorov-Smirnov tests for all metrics (cf. Table 9.4).

	P-Value
Length	$p = 0.1329$
Shannon approx.	$p = 0.5005$
NIST	$p = 0.7400$
Digits	$p = 0.1623$
Upper Chars	$p = 0.7928$
Lower Chars	$p = 0.3494$
Special Chars	$p = 0.6344$
Crackability	$p = 0.4181$

Table 9.5: (Real) Password Metrics for Participants vs. Non-Participants (2-tailed Kolmogorov-Smirnov).

These results suggest that there is no statistically significant difference between both participants and non-participants and hence we believe that our study sample adequately represents our university's population. Summaries of entropy and crackability for both participants and non-participants can be found in Table D.3 and D.4.

9.5 Limitations

This study is limited in several ways.

Population:

Since the ground truth data was drawn from the student population of the university, the study also focused solely on this population. While this offers a certain amount of transferability to similar studies, the results should be used with care when evaluating the behavior of a more diverse population.

Password policies:

Due to the policies in place a certain minimum password quality was enforced. Thus, the range across which participants could behave differently was restricted. Hence, it is possible that different behavior would be more pronounced in unconstrained password creation scenarios. However, since many password systems have policies in place, we believe this to be only a minor limitation in practice.

Self-selection bias:

All participants were self-selected. While this would constitute an ecological validity problem if these results were to be transferred to the general population, we believe in this study it is not a problem, since the matter we are studying (i. e. password studies) usually have the same self-selection procedure and thus results should be accurate in this respect. Additionally, we were able to show that in this case the measured metrics of the passwords of participants and non-participants did not differ significantly (c.f. Table 9.4).

Number of real services:

Not all students were registered for all real services. Consequently we might have missed behavioral patterns that would have become visible if we had been able to analyze more of their passwords. Potentially this could have upgraded a category Single participant to a Full or System.

Study enrollment vs real enrollment:

We expected the participants to enroll in all four services in short succession. While this is not unrealistic per se, the enrollment process at our university does allow students to add services at a later date. There were no logs available to indicate how many students enrolled for all their services when they first signed up and how many added services over time. If students changed the way they choose their passwords between the enrollment for different services, we might have falsely classified a real category Full or System participant as a Single. Four participants also stated that they had felt overburdened by having to choose four passwords in a row.

Changing behavior over time:

The quality of this study could be negatively influenced by a varying amount of time between the last time a participant changed their real password and participation in the study. If a participant genuinely changed the way they create passwords, e. g. adopted the use of a password manager or opted for a different method of designing multiple passwords, we might have misclassified a category Single, Full or System participant as a Null. However, we did not find any significant differences in our ratings based on the age of the user's real passwords. Five participants stated that the reason their study password differed from the real university passwords was due to the fact that they had changed the way the create password in general.

Different Incentives:

We offered online study participants to enter a raffle for three 100 Euro Amazon vouchers, while each lab study participant received 20 Euros immediately. This might have influenced their motivation to put effort into thinking up sensible passwords, which might have contributed to differences in our findings between the two

groups. However, since this mirrors our behavior when conducting real studies, this is an effect we would also encounter in future real studies.

Priming

Due to a technical problem in condition assignment, participants of the online study were not assigned to the priming/non-priming condition in a round robin process but sequentially. We checked for both demographical and study result differences (as discussed in section 9.4) but we did not find any indication that this issue affected our results. A further possible confound is that students assigned to different conditions might have communicated about the study before participating and thus affected the non-priming condition.

Overall, although our dataset is not ideal, we contend that our findings do provide significant insight into the ecological validity of password studies. Since very little is known about this important topic, even imperfect information offers valuable insights at this stage.

9.6 Summary

In this study we presented an empirical analysis on the ecological validity of a password study. We manually compared 645 sets of passwords collected in an online and a laboratory study with real passwords belonging to the same participants for the same kind of services. We classified participants into five categories depending on how closely their study behavior matched their real behavior. We showed that our classification was a good predictor of positive correlation between a number of other password composition metrics as well as a password cracking count produced by John the Ripper. Based on these metrics, we estimate that 29.9% of our participants did not behave as they normally do, while 46.1% percent offered comparable data and 24.0% offered somewhat comparable data. This improves to 19.6%, 57.3% and 23.1% respectively after removing the participants who self-reported that they did not behave normally. To the best of our knowledge, these are the first empirical results on how people's password behavior changes due to the fact that they are participating in a password study.

Take-Aways

- A noteworthy number of study participants (26.5%) used one of their real passwords in the study. Beyond these direct matches, there were many study passwords that were very similar to participants' real passwords. Consequently, passwords gathered during a study should be treated with the same level of protection as real passwords. Normally, we analyze data collected during our studies on our laptops. For this study, we opted to work in encrypted volumes on computers disconnected from the network and all study related data has now been put in an encrypted drive which is stored in a university safe. We will

adopt this procedure for all future password studies, due to the considerable number of participants who used their real passwords during the study.

- While there are participants who do not behave realistically during password studies on the whole, we argue that password studies create useful data to study. However, since real password studies do not know which participants are behaving normally and which are not, more research is needed to find out how to best interpret the results. Great care should be taken when comparing a whole set of study passwords using standard metrics such as password length or NIST since the results can be noticeably skewed by the unrealistic behavior of the Null and Derogatory participants.

- More participants fell into the helpful Single, Full and System categories in our lab condition compared to our online condition. This difference is statistically significant.

- The difference between the priming conditions was minimal. There was no significant difference in our scoring. The slight differences in the NIST entropy were not conclusive.

- In our study, there was a relation between those participants we ranked as Null or Deregatory and those who self-reported they did not behave realistically. While this phenomenon needs to be studied in more detail and with different populations, it seems that adding this kind of self-reporting question to password studies can improve the quality of the data to a certain extent.

- Studies wishing to examine the memorability of passwords need to pay the most attention to ecological validity, since we saw a significant variation between users' normal behavior and their study behavior in respect to writing down passwords and selecting passwords to be memorable only for the duration of the study. Using online studies, participants are able to use all their normal means, i. e. writing passwords down, password managers etc. Conversely, however, a significant number of participants wrote down passwords although they stated they normally don't. The lab condition on the other hand hindered participants who normally wrote down their passwords from doing so. The lab condition also had a significantly higher login failure rate for part two of the study. If brain powered memorability is to be studied, we would recommend a laboratory study over an online study.

This study represents a first step to understanding the effect ecological validity issues have on password studies. There are several important and interesting open questions. One of the most relevant questions for future work is whether MTurkers behave in a similar way to the student population studied in this work. Since we have no ground truth data for MTurkers, other methods for establishing this will have to be found. Another interesting question is how participants behave when not constrained by password policies. While many password systems do use policies, it would nonetheless be interesting to know if there is an additional risk

to the ecological validity of studies that do not use password policy enforcement. Our evaluation of the self-reporting data suggests this is likely to be true. Further progress in terms of ecological validity can be made by optimizing the removal of unsuitable participants using self-reporting data.

10 Conclusions

Throughout this thesis, I demonstrated the importance of usable security and privacy research for four important actors in IT security ecosystems: End users, administrators, developers and system designers are all impacted by the usability of security and privacy mechanisms they use. Additionally I motivated the need for more foundational work on ecologically validity of usable security and privacy research methods.

First, as an important example of end user usable security research, I presented a study on the impact of user interface usability and workflow integration on the acceptance of a message encryption mechanism. My results imply that automating encryption and decryption of messages and not forcing users to interact with the security measure more than strictly necessary – e. g. not having to exchange encryption keys or pressing additional buttons – increases adoption intention. Furthermore, I found that not entirely making the security of the mechanism invisible helped to increase participants' trust in the mechanism. Both results underline the need of reassurance and transparency of security features for users. Importantly, many participants stated that having some sort of a key recovery solution was strictly necessary for an encryption mechanism. This chapter showed that an end user oriented security measure must not impact users' known workflows more than strictly necessary but should also not be entirely invisible to increase users' trust in the security mechanism.

Second, I discussed a study with administrators of HTTPS enabled webservers that deploy a mis-configured TLS certificate. Mis-configured TLS certificates trigger TLS warning messages in browsers, lead to interruptions of users' workflows when surfing the web and force users to unnecessarily interact with a security measure the choices and implications of which they rarely understand. After analyzing certificate configurations gathered by the Google crawler, I conducted interviews with administrators of affected sites to learn the root causes of mis-configurations and to derive possible countermeasures. This chapter has two major contributions: It helps to understand usability issues of the current way administrators have to consider TLS certificate configuration and it helps to estimate the validity of previous research that measured TLS certificate mis-configurations in the wild. While many administrators stated that they had problems to securely configure their TLS certificates due to complexity reasons, other important factors were that not all administrators were aware of the fact that TLS certificates can be obtained for free and that many of the sites that operate mis-configured certificates do not receive much user traffic.

Third, the evaluation of usability issues for developers of password manager apps on Android serves as my introduction to the field of usable security and privacy research targeting information workers. My results and experience from the two previous chapters motivated the idea and design of this study: Based on the assumption that Android app developers' workflows do not leave much room for focusing on security mechanisms (as end users do not think of encryption when they exchange messages online), I designed the study in which I back up self-reported information with real world data. I developed a proof-of-concept exploit to attack all password manager apps in Google Play and found that many apps were vulnerable to my

password sniffing attack that could be prevented by a developer invested in security. I conducted interviews with app developers to learn their motives not to protect their users from this attack vector. The results imply that many app developers were not even aware of a possible security issue and when confronted with a secure solution were overwhelmed by its complexity.

Fourth, I presented an evaluation of the customized implementation of TLS certificate validation in Android apps. Following a similar design as the previous chapter, I used real world Android apps as a motivation and backup for a subsequent developer study in the next chapter. Focusing on the usage of TLS in Android apps instead of password manager apps allowed me to investigate a much larger set of Android apps and to conduct a developer study with more participants to gain more valuable insights. After implementing a static code analysis tool called MalloDroid, I analyzed the top 13,500 Android apps in Google Play and found that 8% of the apps included insecure certificate validation implementations, leaving them vulnerable to Man-In-The-Middle attacks. If found that most insecure solutions implemented a very similar pattern that could be found on developer websites such as `stackoverflow.com`. However, I also found that apps with secure certificate validation oftentimes showed their users confusing or misleading warning messages in the presence of a Man-In-The-Middle attack. Those results are in line with results of the previous chapter: Developers seem to be overwhelmed by security related aspects and prefer quick-and-dirty solutions over implementing secure code.

Fifth, a continuation of Chapter 6 discusses a redesign of how app developers should interact with TLS implementations in apps. Motivated by the static code analysis results for Android, I conducted a similar investigation for the iOS ecosystem to see of Apple's walled garden approach produces more secure solutions. However, my findings imply that as soon as developer have the opportunity to circumvent security mechanisms to quickly produce functional code iOS is not more secure than Android. To learn the root causes for implementing insecure solutions, I conducted interviews with developers of the affected apps. Most developers showed a very limited understanding of the concepts of TLS and mostly were not aware of the implications of their insecure implementations or failed to understand that they undermined the security guarantees offered by TLS. Instead of following a similar approach as in Chapter 5 and encouraging developers to implement a secure solution based on the current API, I developed a different path to make TLS coding in apps more secure and easy to use. I utilized the interview results as input and redesigned the way app developers interact with TLS. In summary, my redesign enforces secure defaults, covers all exceptions I found in real apps and during my developer study and favors configuration over writing code for exotic use-cases such as pinning or self-signed certificates during development. A final evaluation with previous interview candidates proves the efficacy of my solution in terms of security and usability.

Sixth, I presented and discussed Application Transparency, a novel and easy to use mechanism that enables providers of central software repositories to offer cryptographic proofs of software distributions' authenticity and integrity. Central software repositories yield great power and may (be forced to) mount targeted attacks against

specific users of their system. Such targeted attacks may consists of installing malware, withholding important (security) updates or hiding specific software from users. Application Transparency is the first security mechanism, that allows real-time verification of the authentic and integer behaviour of such a repository provider. End users do not have to verify any proofs manually (in contrast to checking hash values). If integrity is given, users are not even aware of the protection mechanism. In an alarming situation, users are made aware of the targeted attack.

Last, I motivated more foundational usable security and privacy research and presented an empirical study on the ecological validity of a password study. I collected 645 sets of passwords in an online and a laboratory study and compared them to sets of real passwords of the same participants. This allowed me to draw conclusions about the participants' behaviour in laboratory and online study setups compared to their real world behaviour. Based on manual analyses and cracking counts produced by John the Ripper, I found that a third of the participants behaved completely different, while less than a half of them provided comparable passwords and another quarter of them provided somewhat comparable data. Removing the participants who self-reported that their study behaviour differed from their real world behaviour, improved those numbers significantly. This work illustrated the importance of basing usable security and privacy research results on ecologically valid study methodologies. Overall, I found that using self-reporting data has to be treated with caution. While most participants reported usable data, a fifth of all participants produced entirely useless results – even after filtering out honest liars. Since a conventional study setup does not easily allow to identify the problematic 20%, this work is a strong indicator that underpins the fact that all usable security and privacy researchers have to review their data collection methodologies very carefully and should ideally back them up with real world information.

10.1 Future Work

Having worked on usable security and privacy topics for end users, administrators, developers and system designers for several years, I can definitely confirm the main hypothesis that motivated my thesis: To improve IT security and privacy measures, most end user facing security mechanisms can benefit from more easy to use tools, APIs and processes for administrators, developers and system designers. Interlocking more usable mechanisms for administrators, developers and system designers significantly increases the potential to result in more usable IT security mechanisms for end users.

My work showed that addressing the underlying security technologies and protocols administrators, developers and system designers work with is a promising direction. In Chapter 3, I demonstrated that although the sensible inclusion of user interface elements can help users when dealing with the encryption of messages, automating key management tasks is a crucial part of improving the usability of a message encryption mechanism. Providing developers an easy to use protocol or API to support them during the implementation of sensible user interfaces with usable

security mechanisms seems to be an important step to help the adoption of message encryption mechanisms. Key management mechanisms based on Google's certificate transparency such as the work by Ryan et al. [170] are promising candidates to be used as building blocks for usable message encryption services.

My extensive work on usability aspects of TLS implementations and configurations for mobile app developers and administrators of HTTPS enabled webservers improved the understanding and assessment of the tremendous usability challenges for end users (cf. Chapters 6, 7, 4). I showed that many developers and administrators lack a profound understanding of TLS and therefore make security decisions – intentionally or accidentally – that result in insecure or unnecessarily complex software for end users. Based on results of my developer study, I was able to address mobile app developers' challenges of TLS implementations that provide both secure defaults – increasing security for end users – and easy to use concepts to work around those defaults in a secure and usable way. Interesting applications for future work are the usability of cryptographic APIs as addressed by libsodium [139] or TLS certificate deployment mechanisms as provided by letsencrypt [102]. Furthermore, helping the adoption of research results by the IT industry is an interesting aspect of future work in this area.

Working on Application Transparency allowed me to gain insights into the complex process of distributing software and its threats for end users. Figuring out that verifying the authenticity and integrity of software is an almost insurmountable challenge for end users makes it very important to continue working in this field. Solutions based on Application Transparency or similar mechanisms seem promising candidates to solve this problem. Providing strong cryptographic proofs that are easily verifiable during the installation process can be building blocks for usable mechanisms for end users, information workers and market providers.

Finally, I evaluated the ecological validity of two specific laboratory and online study designs to investigate passwords in Chapter 9 and derived important insights abouts the practice of collecting passwords in a user study. This study is intended to be an example of how more foundational usable security and privacy research should look like and motivate the need for more ecological validity studies to deduce generally applicable rules for user studies. Interesting areas of research are warning message and permission studies, which are typically based on online studies and the self-reporting of participants. A better understanding of whether those study designs produce ecologically valid results may significantly help to improve future research. Ecological validity studies in the context of administrators, developers and system designers can offer valuable input for future research.

A Appendix: Message Encryption Study

Note: The following questions use colloquial language on purpose to create a comfortable atmosphere. It was also translated from German for inclusion in this thesis.

A.1 Questionnaire Items

Pre-Test

For how long have you been using Facebook?
Choose one answer: For 1 month, For 6 months, For 1 year, For 2 years, Longer, I don't know, n/a.

How often have you forgotten your Facebook password in the last 12 months?
Choose one answer: Never, Once, Twice, Three times, More than three times, n/a, I don't know, Other.

How important is it to you that only you and the recipient can read private messages?
Rate from 1 (unimportant) to 5 (important).

How often do you use Facebook on average?
Choose one answer: Less than an hour per day, 1 to 2 hours per day, 2 to 4 hours per day, More than 4 hours per day, More than once per week, Once per week, Monthly, Less frequently than once per month, n/a.

How many friends do you approximately have on Facebook?

How many Facebook messages do you send per week on average?

How many of these messages have more than one recipient?

How many of these messages do you consider worthy of protection?

How often do you use the chat feature on Facebook?
Choose one answer: More than once per day, On a daily basis, On a weekly basis, Less than once per week, Never, n/a.

How easy do you think it is for the following persons or organisations to read your private messages on Facebook?
Rate from 1 (very easy) to 5 (very hard) for the following: Friends, Hackers, Facebook employees, Advertising Companies, US government, German government.

How high do you think is the motivation for the following persons or organisations to read your private messages on Facebook?
Rate from 1 (very low) to 5 (very high) for the following: Friends, Hackers, Facebook employees, Advertising Companies, US government, German government.

How much would it concern you if the following persons or organisations were able to read your private messages on Facebook?
Rate from 1 (very little) to 5 (very much) for the following: Friends, Hackers, Facebook employees, Advertising Companies, US government, German government.

How well do you feel you and your privacy are protected when communicating through Facebook messages?
Choose from 1 (not at all) to 5 (very well).

How well do you feel you and your privacy are protected when communicating through Facebook chat?
Choose from 1 (not at all) to 5 (very well).

Post-Task

Please rate the following questions regarding the mechanism you just used.
Choose from 1 (strongly disagree) to 5 (strongly agree) for the following:

1. I think that I would like to use this system frequently;
2. I found the system unnecessarily complex;
3. I thought the system was easy to use;
4. I think I would need the support of a technical person to be able to use the system;
5. I found the various functions in this system well integrated;
6. I thought this system was too inconsistent;

7. I would imagine that most people could learn to use this system very quickly;
8. I found the system very cumbersome to use; I felt very confident using the system;
9. I needed to learn a lot of things before I could get going with this system.

Please rate the following questions regarding the mechanism you just used.
Choose from 1 (strongly disagree) to 5 (strongly agree) for the following:

1. I would send private messages using this mechanism in the future;
2. I would send all my messages using this mechanism in the future;
3. I feel that my messages are now well protected.

Final

Please enter your age.

Please specify your gender.

Please enter your major subject.

A password is needed to use an encryption mechanism. If losing or forgetting the password led to the loss of all previous private messages, would you use such an encryption mechanism?
Choose yes or no.

Please rate the following statements with regard to the previous question about password recovery.
Choose from 1 (strongly agree) to 5 (strongly disagree) for the following: I am worried about forgetting my password; I am worried about the potential loss of all my previous messages.

Would you prefer a mechanism that is able to recover your password like it is possible on the Facebook website?
Choose yes or no.

Do you use software to encrypt your data?
Choose one or more answers: Yes, for Facebook; Yes, for email; Yes, for my hard disk; I don't know; No; Yes, for: ...

When friends have computer problems, they often ask me for help.
Choose from 1 (strongly disagree) to 5 (strongly agree).

When I have computer problems, I often ask my friends for help.
Choose from 1 (strongly disagree) to 5 (strongly agree).

What is AES?
Choose one or more answers: A browser extension; A Facebook application to store images; An encryption mechanism; I don't know; Something else: ...

Do you have any comments on this study, the procedure, the technologies used or anything else? Free Text

A.2 Interview Guideline

The following gives a brief overview of the questions I asked in the semi-structured interview in the final study.

FBMCrypt Account

Please rate the effort for creating a FBMCrypt account.
With respect to the application, please rate the appropriateness of creating an extra account for encrypting Facebook messages.
Please compare the account creation process with the creation of a new Facebook and webmail account.
Please comment on the fact that your FBMCrypt password had to be different from your Facebook password.
How likely would it be that you forget your FBMCrypt password?
Have you ever forgotten a password? Your Facebook password?
Please attribute the FBMCrypt-to-Facebook account binding process.
Please comment on the plugin installation procedure.

Facebook Messaging

How many private Facebook messages do you send per week?
In your opinion, do these comprise sensitive information? If not, what channel do you use to transport sensitive information? If yes, what is the amount of sensible messages?
Do you have reservations that an unauthorised third party could access your private Facebook messages? If yes, who do you think is able to do so? If not, why do you think your messages are secure?

FBMCrypt Workflow

Please comment on the process of sending a FBMCrypt-protected private Facebook message.

Please describe the message composer you used to send a FBMCrypt-protected private Facebook message.
Please describe the message composer you used to send a private Facebook message that was not encrypted.
Please describe reading a FBMCrypt-protected private Facebook message.
Please describe the presentation of a FBMCrypt-protected private Facebook message when reading it.

Satisfaction/Perceived Security

Would you send all your private Facebook messages using the FBMCrypt service? If not, why and which messages would you not send using FBMCrypt?
Please compare your perceived feeling of security sending a private Facebook message the normal way to sending a message with FBMCrypt.
Would you recommend FBMCrypt to your friends?
Would you pay for the FBMCrypt service? If yes, how much would you be willing to pay?

Key Recovery

Would you use FBMCrypt if losing or forgetting the password would result in losing access to your private Facebook messages? If yes/no, why?
Would you prefer a mechanism that allows for recovery of the encryption password?

B Appendix: Webmaster Study

B.1 Contact Email

Dear Webmaster,

we are an IT security research group at Leibniz University Hannover, Germany conducting a study on the use of X.509 certificates on websites.

We are contacting you since we found that your website https://www.example.com is operating an X.509 certificate which triggers an TLS warning message for users visiting your website.

Since this is a common issue, we are attempting to identify the causes and see what needs to be done to improve the usability of TLS and certificate configuration for administrators.

We would very much appreciate if you could give us 5-7 minutes of your time and answer a short survey either by replying to this email or via an online survey (https://www.our-uni.com/survey) - which ever you prefer, although the online survey is easier. We will not ask any privacy related questions and all answers will be evaluated anonymously and in such a way that no link to you or your website can be made.

If you have any questions or comments related to our study please do not hesitate to contact Sascha Fahl <fahl@dcsec.uni-hannover.de> or visit https://www.dcsec.uni-hannover.de/ssl-study.

We would appreciate your participation and contribution to our study very much.

As explained in the email, the survey could either be answered by replying to our email or clicking a link to an online survey we hosted.

C Appendix: Studying Android's TLS Warning Message

C.1 Online Survey

We based the questions of our online survey on previous surveys [184, 183, 160], adapting them to our scenario and optimizing the survey for mobile delivery. For this purpose, we removed most of the free text answers and replaced them by multiple choice or radio button answers to make the online survey easier to handle on an Android smartphone.

After clicking a link on the landing page to begin the study, participants were redirected to a non-university domain with a page designed to look like Android's default browser warning message. The warning message was interactive, hence users could click on "Certificate Details" for more information. The page thus replicated the user experience of a real TLS warning message in Android's default browser.

We presented two different TLS warnings, although, just as with the real Android TLS warnings, the difference only became visible if the user clicked on "Certificate Details". One warning stated that the certificate was signed by an untrusted CA and the other warning stated that the hostname did not match the certificate's common name.

We tracked whether the participants clicked "Continue" or "Cancel". In both cases, participants were directed to the first page of the questionnaire that explained that the message just shown was part of the study. For half of the participants, the study was served via HTTPS, and for the other half, it was served via plain HTTP. Hence, we had four different groups: `untrustedCA+HTTP`, `untrustedCA+HTTPS`, `wronghostname+HTTP` and `wronghostname+HTTPS`. The survey was also hosted on a domain that did not obviously belong to our universities, in order to avoid the implicit trust often associated with university servers. Unlike previous studies ([184], [183] and [84]), we did not refer to the TLS warning message as a warning message during the online survey. Instead, we called it a popup message to use a neutral term avoiding a bias in the users' perceptions. Subsequently, questions contained in

the online survey are listed. In addition to TLS warning message comprehension, HTTPS indicator comprehension, Android usage and online security awareness, we asked the participants about their self-reported technical expertise and demographic information. Due to space constraints, questions from the last two categories are not listed below.

TLS Warning Message Comprehension

- The popup message you just saw is part of this survey. Have you previously seen this kind of message while surfing the Internet with your Android phone?
  - (Yes, No, I'm not sure)
- Did you read the entire text of the popup message?
  - (Yes, Only partially, No)
- Please rate the following statements (all statements were rated on a 5-point Likert scale, ranging from "Don't agree" to "Totally agree"):
  - I always read these kind of popup messages entirely.
  - I understood the popup message.
  - I am not interested in such popup messages.
  - I already knew this popup message.
  - I am only interested in winning the voucher.
- When you saw the popup message, what was your first reaction?
  - I was thankful for the message.
  - I was annoyed by the popup.
  - I didn't care.
  - Other: (text field)
- Please rate the amount of risk you feel you were warned against.
  - 5-point Likert scale ranging from "Very low risk" to "Very high risk"
- What action, if any, did the popup message want you to take?
  - To not continue to the website.
  - To be careful while continuing to the website.
  - To continue to the website.
  - I did not feel it wanted me to take any action.
  - Other: (text field)
- How much did the following factors influence your decision to heed or ignore the popup message? (all factors were rated on a 5-point Likert scale, ranging from "Very little influence" to "Very high influence")

- The text of the message.
- The colors of the message.
- The choices that the message presented.
- The destination URL.
- The chance to win a voucher.
- The fact that this is an online survey.
- Other factors: (text field)

- Which factor had the most influence on your decision?
  - The text of the message.
  - The colors of the message.
  - The choices that the message presented.
  - The destination URL.
  - The chance to win a voucher.
  - The fact that this is an online survey.
  - Other factors: (text field)

HTTPS Indicator Comprehension

- Is the Internet connection to this online survey secure?
  - (Yes, No, I'm not sure)
- Please explain your decision:
  if answered with "yes"
  - I trust my service provider.
  - I trust my smartphone.
  - The URL starts with `https://`.
  - All Internet communication is secure.
  - A lock symbol is visible in the browser bar.
  - Other: (text field)

  if answered "no"
  - I do not trust my service provider.
  - I do not trust my smartphone.
  - The URL starts with `http://`
  - Communicating over the Internet is always insecure.
  - There is no lock symbol in the browser bar.
  - The address bar is not green.

- Other: (text field)

if answered "don't know"

- I don't know how to determine this.
- I don't care.
- I don't trust the visual indicators.
- I don't trust IT in general.
- Other: (text field)

Android Usage

- For how long have you been using an Android smartphone?
  - 1 month or less
  - 2 - 6 months
  - 7 - 11 months
  - 1 - 2 years
  - more than 2 years
- Did you turn off browser warning messages?
- How many apps have you installed on your phone?

Online Security Awareness

- Have you ever had any online account information stolen? (Yes, No)
- Have you ever found fraudulent transactions on a bank statement? (Yes, No)
- Have you ever been notified that your personal information has been stolen or compromised? (Yes, No)
- Have you ever lost your smartphone? (Yes, No)

D Appendix: Ecological Validity of a Password Study

Note: The following questions use colloquial language on purpose to create a comfortable atmosphere. It was translated from German for inclusion in this thesis.

D.1 Question Plan

We asked our participants to self-report several aspects of their password usage behavior using the following questions (translated from German):

Which usage behavior concerning passwords for Internet services best mirrors your behavior?
Please select one of the following answers: I use the same password for all of my accounts.; I use between 2 and 5 different passwords for all my accounts.; I use between 6 and 10 different passwords for all my accounts.; Each of my accounts has a unique password.; Other

Please specify how you keep track of your passwords.
Please select all appropriate answers. I memorize all my passwords.; I came up with a scheme that allows me to deduce the password for the respective service whenever needed.; I wrote my passwords onto a piece of paper stored in a safe place that I consult when needed.; I use a password manager that stores my usernames and passwords for me.

Please select the appropriate answer for each statement.
Rate your agreement from "I agree completely" (1) to "I disagree completely" with the following statements: The passwords I created are similar to my real passwords.; I chose a completely different type of password than I normally would.; The passwords I created are less secure than my real passwords.; The passwords I created

are more secure than my real passwords.

D.2 Contingency Tables

	Derogatory		**Null**		**Single**		**Full**		**System**	
	real	study	real	study	real	study	real	study	real	study
Length	0.2857	0.9883	0.6353	0.9145	0.1373	0.0854	0.4663	0.4859	0.6160	0.9132
Shannon	0.7460	0.9683	0.2639	0.4083	0.1072	0.1521	0.5290	0.7270	0.9445	0.7264
NIST	0.1641	0.9483	0.8775	0.9934	0.0117	0.5886	0.8292	0.6100	0.9445	0.9445
Digits	0.9483	0.9483	0.4571	0.7415	0.1394	0.9529	0.7030	0.4342	0.2177	1.0000
Upper Chars	0.4005	0.9883	0.7442	0.9993	0.7683	0.4521	0.9996	0.7453	0.9680	0.9445
Lower Chars	0.5121	1.0000	0.9163	0.9091	0.3403	0.1865	0.1329	0.6774	0.6714	0.9838
Special Chars	0.5121	0.9991	0.1412	0.9988	0.1587	0.3093	0.3598	0.7514	0.1892	0.5081
Crackability	1.0000	0.9991	0.6663	0.9999	1.0000	0.9371	1.0000	0.9848	1.0000	0.9838

Table D.1: Priming vs. Non-Priming (2-tailed Kolmogorov-Smirnov; P-Values).

We conducted a two-tailed Kolmogorov-Smirnov Test, the null hypothesis being that the priming and non-priming password sets were from the same population concerning the metrics above. Since we could not find statistically significant differences between the priming and non-priming groups, we concluded that priming did not have significant effects on our subjects within the respective categories. This enabled us to evaluate the effect of the type of study solely on the number of password sets we scored into the respective categories.

	Null		**Single**		**Full**		**System**	
	real	study	real	study	real	study	real	study
Length	0.6878	0.7523	0.8868	0.5431	0.3741	0.4377	0.9972	0.9039
Shannon	0.4551	0.7204	0.4890	0.6154	0.4624	0.3556	0.8727	0.8727
NIST	0.3550	0.9942	0.4304	0.4519	0.1509	0.8704	0.6734	0.6734
Digits	0.5154	0.9718	0.9996	0.4234	0.3458	0.4092	0.2770	0.9906
Upper Chars	0.9930	0.6332	0.6931	0.1710	0.8282	0.8236	0.5441	1.0000
Lower Chars	0.8680	0.4649	0.9444	0.2381	0.0435	0.2871	0.9972	0.8888
Special Chars	0.9598	0.9275	0.9119	0.9997	0.8645	1.0000	0.4435	0.4435
Crackability	0.6769	0.8034	0.9999	0.9315	0.9950	0.9863	0.7994	0.7994

Table D.2: Lab vs. Online (2-tailed Kolmogorov-Smirnov; P-Values).

We conducted a two-tailed Kolmogorov-Smirnov Test, the null hypothesis being that the password sets from the lab and the online participants in each category were from the same population concerning the metrics above. Since we could not find statistically significant differences between lab and online participants, we believe that our manual scoring was consistent irrespective of the type of study. This enabled us to evaluate the effect of the type of study solely on the number of password sets we scored into the respective categories.

Table D.3: Entropy and Crackability Summaries for all Passwords of Participants

	Real						
	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Shapiro-Wilk
Shannon	47.26	55.57	62.52	63.64	69.79	99.79	$p < 0.0005$
NIST	18.00	25.50	30.75	29.77	33.50	42.00	$p < 0.0005$
Crackability	0.00%	0.0%	0.0%	16.82%	25.00%	100.00%	$p < 0.0005$
	Study						
	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Shapiro-Wilk
Shannon	47.26	56.02	63.88	64.87	71.45	102.80	$p < 0.0005$
NIST	24.00	30.75	32.63	32.86	34.50	42.00	$p < 0.0005$
Crackability	0.00%	0.0%	0.0%	15.47%	25.00%	100.00%	$p < 0.0005$

Table D.4: Entropy and Crackability Summaries for all Passwords of Non-Participants

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
Shannon	47.45	56.56	63.51	64.43	71.28	103.10
NIST	24.00	30.75	32.50	32.64	34.50	42.00
Crackability	0.00%	0.00%	0.00%	17.87%	33.00%	100.00%

Table D.5: Contingency - Table Self-Reported Different Password Behavior

	Self-Reporting (FET (alt. = greater) p<0.0005)		
Category	Different	Non-Different	Total
Unhelpful	109	84	193
Helpful	148	304	452
Total by Self-Reporting	257	388	645

Table D.6: Contingency Table - Study Completion by Scoring

	Study Completion (FET (alt. = two-tailed) p=0.9166)		
Category	Completed	Did-Not-Complete	Total
Unhelpful	151	42	193
Helpful	356	96	452
Total by Completeness	507	138	645

Table D.7: Contingency Table - Password Strategy by Scoring

	Password Strategy (FET (alt. = greater) p=0.01253)		
Category	Individual-Passwords	No-Individual-Passwords	Total
Unhelpful	39	154	193
Helpful	57	395	452
Total by Strategy	96	549	645

Table D.8: Contingency Table - Re-Login Rate by Scoring

	Re-Login (FET (alt. = two-tailed) p=0.6063)		
Category	Login-Success	Login-Failure	Total
Unhelpful	165	28	193
Helpful	81	371	452
Total by Login Success	246	399	645

Table D.9: Contingency Table - Scoring Results Online/Lab

	Scoring Results (FET (alternative=greater) p=0.0296)		
Category	Online	Lab	Total
Unhelpful	181	12	193
Helpful	401	51	452
Total by Study Type	582	63	645

	Study-Real					
	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
Length	0.00	0.25	0.50	0.13	0.25	0.00
Digits	0.00	−0.50	−0.17	−0.29	0.00	−0.25
Upper Chars	0.00	0.33	0.00	0.17	0.33	2.50
Lower Chars	0.00	0.00	0.00	0.20	0.42	0.75
Special Chars	0.00	0.00	−0.08	0.03	0.00	0.50
Shannon	0.00	0.45	1.40	1.23	1.66	3.01
NIST	6.00	5.25	1.90	3.09	1.00	0.00

Table D.10: Metric-Delta Summaries for all Passwords

Bibliography

[1] Yasemin Acar, Michael Backes, Sven Bugiel, Sascha Fahl, Patrick McDaniel, and Matthew Smith. Sok: Lessons learned from android security research for appified software platforms. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2016.

[2] Anne Adams and Martina Angela Sasse. Users are not the enemy. *Communications of the ACM*, 1999.

[3] Devdatta Akhawe, Bernhard Amann, Matthias Vallentin, and Robin Sommer. Here's my cert, so trust me, maybe?: Understanding tls errors on the web. In *ACM WWW*, 2013.

[4] Devdatta Akhawe and Adrienne Porter Felt. Alice in warningland: A large-scale field study of browser security warning effectiveness. In *Proceedings of the 22nd USENIX Security Symposium*, 2013.

[5] Deena Alghamdi, Ivan Flechais, and Marina Jirotka. Security Practices for Households Bank Customers in the Kingdom of Saudi Arabia. *Eleventh Symposium On Usable Privacy and Security (SOUPS 2015)*, 2015.

[6] Jonathan Anderson, Claudia Diaz, Joseph Bonneau, and Frank Stajano. Privacy-enabling social networking over untrusted networks. In *Proceedings of the 2nd ACM Workshop on Online Social Networks*, pages 1–6, 2009.

[7] Sarah J Andrabi, Michael K Reiter, and Cynthia Sturton. Usability of Augmented Reality for Revealing Secret Messages to Users but Not Their Devices. *2015 Symposium on Usable Privacy and Security, USENIX*, 2015.

[8] Android. Android dashboard. `http://developer.android.com/about/dashboards/index.html`, January 2014.

[9] Erinn Atwater, Cecylia Bocovich, Urs Hengartner, Ed Lank, and Ian Goldberg. Leading Johnny to Water: Designing for Usability and Trust. *Eleventh Symposium On Usable Privacy and Security (SOUPS 2015)*, 2015.

[10] Kathy Wain Yee Au, Yi Fan Zhou, Zhen Huang, and David Lie. Pscout: analyzing the android permission specification. In *CCS12*. ACM, 2012.

[11] Randy Baden, Adam Bender, Neil Spring, Bobby Bhattacharjee, and Daniel Starin. Persona: An online social network with user-defined privacy. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, pages 135–146, 2009.

[12] David Barrera, Jeremy Clark, Daniel McCarney, and Paul C. van Oorschot. Understanding and improving app installation security mechanisms through empirical analysis of android. In *Proc. 2nd Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM '12)*. ACM, 2012.

[13] Adam Bates, Joe Pletcher, Tyler Nichols, Braden Hollembaek, Dave Tian, Kevin R.B. Butler, and Abdulrahman Alkhelaifi. Securing ssl certificate verification through dynamic linking. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 394–405, New York, NY, USA, 2014. ACM.

[14] Filipe Beato, Markulf Kohlweiss, and Karel Wouters. Scramble! your social network data. In *Proceedings of the 11th International Conference on Privacy Enhancing Technologies*, pages 211–225. Springer, 2011.

[15] Benjamin Beurdouche, Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Jean Karim Zinzindohoue. A messy state of the union: Taming the composite state machines of tls. In *IEEE Symposium on Security and Privacy*, pages 535–552. IEEE Computer Society, 2015.

[16] Robert Biddle, Sonia Chiasson, and P.C. Van Oorschot. Graphical passwords: Learning from the first twelve years. *ACM Comput. Surv.*, 44(4):19:1–19:41, September 2012.

[17] M Bishop and D V Klein. Improving system security via proactive password checking. *Computers & Security*, 14(3), 1995.

[18] Bluebox BlackHat. Android master key - bluebox black hat talk. `https://media.blackhat.com/us-13/US-13-Forristal-Android-One-Root-to-Own-Them/-All-Slides.pdf`, January 2014.

[19] Simon Blake-Wilson, Magnus Nystrom, David Hopwood, Jan Mikkelsen, and Tim Wright. Transport Layer Security (TLS) Extensions. Internet RFC 3546, June 2003.

[20] Bluebox. Android master key - bluebox. `http://bluebox.com/corporate-blog/bluebox-uncovers-android-master-key/`, January 2014.

[21] Joseph Bonneau. The science of guessing: Analyzing an anonymized corpus of 70 million passwords. *Security and Privacy (SP), 2012 IEEE Symposium on*, 2012.

[22] Joseph Bonneau, Cormac Herley, Paul C. van Oorschot, and Frank Stajano. Passwords and the evolution of imperfect authentication. *Commun. ACM*, 58(7):78–87, June 2015.

[23] Joseph Bonneau, Cormarc Herley, Paul C. van Oorschot, and Frank Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *Proc. IEEE S&P*, 2012.

[24] Cristian Bravo-Lillo, Lorrie Cranor, Julie Downs, Saranga Komanduri, Stuart Schechter, and Manya Sleeper. Operating system framed in case of mistaken identity: Measuring the success of web-based spoofing attacks on os password-entry dialogs. In *Proc. ACM CCS*, 2012.

[25] Cristian Bravo-Lillo, Lorrie Faith Cranor, Julie Downs, and Saranga Komanduri. Bridging the gap in computer security warnings: A mental model approach. *IEEE Security and Privacy*, 9(2):18–26, March 2011.

[26] Cristian Bravo-Lillo, Lorrie Faith Cranor, Julie S. Downs, Saranga Komanduri, and Manya Sleeper. Improving computer security dialogs. In *Human-Computer Interaction - INTERACT 2011 - 13th IFIP TC 13 International Conference, Lisbon, Portugal, September 5-9, 2011, Proceedings, Part IV*, pages 18–35, 2011.

[27] Cristian Bravo-Lillo, Saranga Komanduri, Lorrie Faith Cranor, Robert W. Reeder, Manya Sleeper, Julie Downs, and Stuart Schechter. Your attention please: Designing security-decision uis to make genuine risks harder to ignore. In *Proceedings of the Ninth Symposium on Usable Privacy and Security*, SOUPS '13, pages 6:1–6:12, New York, NY, USA, 2013. ACM.

[28] J. Brooke. SUS: A quick and dirty usability scale. In P. W. Jordan, B. Weerdmeester, A. Thomas, and I. L. Mclelland, editors, *Usability evaluation in industry*. Taylor and Francis, London, 1996.

[29] John Brooke. Sus: A "quick and dirty" usability scale. In P.W. Jordan, B. Thomas, B.A. Weerdmeester, and A.L. McClelland, editors, *Usability Evaluation in Industry*. Taylor and Francis, 1996.

[30] Chad Brubaker, Suman Jana, Baishakhi Ray, Sarfraz Khurshid, and Vitaly Shmatikov. Using frankencerts for automated adversarial testing of certificate validation in ssl/tls implementations. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, SP '14, pages 114–129, Washington, DC, USA, 2014. IEEE Computer Society.

[31] Sven Bugiel, Lucas Davi, Alexandra Dmitrienko, Thomas Fischer, Ahmad-Reza Sadeghi, and Bhargava Shastry. Towards taming privilege-escalation attacks on android. In *Proceedings of the 19th Network and Distributed System Security Symposium*, 2012.

[32] Michael Buhrmester, Tracy Kwang, and Samuel D. Gosling. Amazon's mechanical turk: A new source of inexpensive, yet high-quality, data? *Perspectives on Psychological Science*, 6(1):3–5, feb 2011.

[33] Chris Callison-Burch. Fast, cheap, and creative: Evaluating translation quality using amazon's mechanical turk. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1 - Volume 1*, EMNLP '09, pages 286–295, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[34] Patrick P.F. Chan, Lucas C.K. Hui, and S.M. Yiu. Droidchecker: Analyzing android applications for capability leaks. In *WISEC '12: Proceedings of the Fifth ACM Conference on Security and Privacy in Wireless and Mobile Networks*. ACM Press, April 2012.

[35] Farah Chanchary and Sonia Chiasson. User Perceptions of Sharing, Advertising, and Tracking. *Symposium on Usable Privacy and Security (SOUPS) 2015, July 22–24*, 2015.

[36] Stephen Checkoway, Ruben Niederhagen, Adam Everspaugh, Matthew Green, Tanja Lange, Thomas Ristenpart, Daniel J. Bernstein, Jake Maskiewicz, Hovav Shacham, and Matthew Fredrikson. On the practical exploitability of dual ec in tls implementations. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 319–335, San Diego, CA, August 2014. USENIX Association.

[37] Ivan Cherapau, Ildar Muslukhov, Nalin Asanka, and Konstantin Beznosov. On the Impact of Touch ID on iPhone Passcodes. *Eleventh Symposium On Usable Privacy and Security (SOUPS 2015)*, pages 257–276, 2015.

[38] William Cheswick. Rethinking passwords. *Commun. ACM*, 56(2):40–44, February 2013.

[39] Pern Hui Chia, Yusuke Yamamoto, and N. Asokan. Is this app safe?: A large scale study on application permissions and risk signals. In *Proceedings of the 2012 Conference on World Wide Web*, WWW '12, 2012.

[40] Sonia Chiasson, Robert Biddle, and P 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*. ACM, jul 2007.

[41] Sonia Chiasson, Paul C. van Oorschot, and Robert Biddle. A usability study and critique of two password managers. In *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*, USENIX-SS'06, Berkeley, CA, USA, 2006. USENIX Association.

[42] Erika Chin, Adrienne Porter Felt, Kate Greenwood, and David Wagner. Analyzing inter-application communication in Android. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*. ACM, 2011.

[43] Erika Chin, Adrienne Porter Felt, Vyas Sekar, and David Wagner. Measuring user confidence in smartphone security and privacy. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*. ACM, 2012.

[44] Jeremy Clark and Paul C. van Oorschot. Sok: SSL and HTTPS: revisiting past challenges and evaluating certificate trust model enhancements. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 511–525, 2013.

[45] D. Crocker. Mailbox names for common services, roles and functions. RFC 2142 (Proposed Standard), May 1997.

[46] Alexei Czeskis, Michael Dietz, Tadayoshi Kohno, Dan Wallach, and Dirk Balfanz. Strengthening user authentication through opportunistic cryptographic identity assertions. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 404–414, New York, NY, USA, 2012. ACM.

[47] et al. D. Cooper. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. `http://tools.ietf.org/html/rfc5280`, May 2008.

[48] Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov, and XiaoFeng Wang. The tangled web of password reuse. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*, 2014.

[49] Lucas Davi, Alexandra Dmitrienko, Ahmad-Reza Sadeghi, and Marcel Winandy. Privilege escalation attacks on android. In *Proceedings of the 13th International Conference on Information Security*, pages 346–360. Springer, 2010.

[50] Mattea Dell'Amico, Pietro Michiardi, and Yves Roudier. Password strength: An empirical analysis. In *INFOCOM, 2010 Proceedings IEEE*, 2010.

[51] Nico d'Heureuse, Felipe Huici, Mayutan Arumaithurai, Mohamed Ahmed, Konstantina Papagiannaki, and Saverio Niccolini. What's app?: A wide-scale measurement study of smart phone markets. *SIGMOBILE Mob. Comput. Commun. Rev.*, nov 2012.

[52] Thomas Dierks and Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176.

[53] DigiCert. Digicert - certificate transparency. http://www.digicert.com/news/2013-09-24-certificate-transparency.htm, September 2013.

[54] Ben Dodson, Ian Vo, T. J. Purtell, Aemon Cannon, and Monica S. Lam. Musubi: Disintermediated interactive social feeds for mobile devices. In *Proceedings of the 21st International Conference on World Wide Web*, pages 211 – 220, 2012.

[55] Stewart I. Donaldson and Elisa J. Grant-Vallone. Understanding self-report bias in organizational behavior research. *Journal of Business and Psychology*, 17(2):245–260, December 2002.

[56] Bryan Dosono, Jordan Hayes, and Yang Wang. " I'm Stuck !": A Contextual Inquiry of People with Visual Impairments in Authentication. *Proceedings of the eleventh Symposium On Usable Privacy and Security*, pages 151–168, 2015.

[57] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. ZMap: Fast Internet-wide scanning and its security applications. In *22nd USENIX Security Symposium*, August 2013.

[58] Peter Eckersley. Sovereign key cryptography for internet domains. https://git.eff.org/?p=sovereign-keys.git;a=blob;f=sovereign-key-design.txt;hb=HEAD, 2011.

[59] 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*. ACM, 2013.

[60] S. Egelman, L.F. Cranor, and J. Hong. You've Been Warned: An Empirical Study of the Effectiveness of Web Browser Phishing Warnings. In *Proceedings of the 26th Annual SIGCHI Conference on Human Factors in Computing Systems*, pages 1065–1074, 2008.

[61] 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*, CHI '13, pages 2379–2388, New York, NY, USA, 2013. ACM.

[62] Serge Egelman, Janice Tsai, Lorrie Faith Cranor, and Alessandro Acquisti. Timing is everything?: The effects of timing and placement of online privacy indicators. In *Proceedings of the 27th International Conference on Human Factors in Computing Systems*, pages 319–328. ACM, 2009.

[63] André Egners, Björn Marschollek, and Ulrike Meyer. Messing with android's permission model. In *Proceedings of the IEEE TrustCom*, pages 1–22, apr 2012.

[64] Andre Egners, Ulrike Meyer, and Bjorn Marschollek. Messing with Android's Permission Model. In *TRUSTCOM '12: Proceedings of the 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*. IEEE Computer Society, June 2012.

[65] Eldad Eilam and Elliot J. Chikofsky. *Reversing : secrets of reverse engineering*. Wiley, Indianapolis (Ind.), 2005.

[66] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*. USENIX Association, oct 2010.

[67] William Enck, Damien Octeau, Patrick McDaniel, and Swarat Chaudhuri. A study of android application security. In *Proceedings of the 20th USENIX Conference on Security*. USENIX, aug 2011.

[68] William Enck, M Ongtang, and P McDaniel. Understanding android security. In *Proceedings of the IEEE International Conference on Security & Privacy*, pages 50–57, 2009.

[69] William Enck, Machigar Ongtang, and Patrick McDaniel. On lightweight mobile phone application certification. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, nov 2009.

[70] M. Myers et al. X.509 Internet Public Key Infrastructure, Online Certificate Status Protocol - OCSP. http://tools.ietf.org/html/rfc2560, June 1999.

[71] C Evans, C Palmer, and R Sleevi. Public Key Pinning Extension for HTTP. https://tools.ietf.org/html/rfc7469, April 2015.

[72] Facebook. The current state of smtp starttls deployment. https://www.facebook.com/notes/protect-the-graph/the-current-state-of-smtp-starttls-deployment/1453015901605223/.

[73] 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*, ASIA CCS '14, pages 507–512, New York, NY, USA, 2014. ACM.

[74] Sascha Fahl, Sergej Dechand, Henning Perl, Felix Fischer, Jaromir Smrcek, and Matthew Smith. Hey, nsa: Stay away from my market! future proofing app markets against powerful attackers. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 1143–1155, New York, NY, USA, 2014. ACM.

[75] 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*, SOUPS '13, pages 13:1–13:13, New York, NY, USA, 2013. ACM.

[76] 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*, CCS '12, pages 50–61, New York, NY, USA, 2012. ACM.

[77] Sascha Fahl, Marian Harbach, Thomas Muders, and Matthew Smith. Confidentiality as a service - usable security for the cloud. In *Proceedings of the IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, 2012.

[78] Sascha Fahl, Marian Harbach, Thomas Muders, and Matthew Smith. Trustsplit: Usable confidentiality for social network messaging. In *Proceedings of the ACM Conference on Hypertext and Hypermedia*, 2012.

[79] 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*, SOUPS '12, pages 11:1–11:17, New York, NY, USA, 2012. ACM.

[80] Sascha Fahl, Marian Harbach, Marten Oltrogge, Thomas Muders, and Matthew Smith. Hey, you, get off of my clipboard - on how usability trumps security in android password managers. In *Financial Cryptography*, Lecture Notes in Computer Science. Springer, 2013.

[81] 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*, CCS '13. ACM, 2013.

[82] Adrienne Porter Felt, Alex Ainslie, Robert W. Reeder, Sunny Consolvo, Somas Thyagaraja, Alan Bettes, Helen Harris, and Jeff Grimes. Improving ssl warnings: Comprehension and adherence. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 2893–2902, New York, NY, USA, 2015. ACM.

[83] Adrienne Porter Felt, Serge Egelman, Matthew Finifter, Devdatta Akhawe, and David Wagner. How to ask for permission. In *Proceedings of the 7th USENIX conference on Hot Topics in Security*, 2012.

[84] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. Android permissions: User attention, comprehension, and behavior. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, SOUPS '12, 2012.

[85] Andy Field, Jeremy Miles, and Zoe Field. *Discovering Statistics Using R.* SAGE Publications, 2012.

[86] R. Fielding and J. Reschke. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. http://tools.ietf.org/html/rfc7230, June 2014.

[87] D Florencio and C Herley. A large-scale study of web password habits. *Proceedings of the 16th international conference on World Wide Web*, 2007.

[88] 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*. ACM, jul 2008.

[89] S. Frankel and S. Krishnan. IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap. IETF RFC 6071, February 2011.

[90] A Freier, P Karlton, and P Kocher. The Secure Sockets Layer (SSL) Protocol Version 3.0. https://www.ietf.org/rfc/rfc6101.txt, August 2011.

[91] Simson Garfinkel and Heather Richter Lipford. *Usable Security: History, Themes, and Challenges*. Synthesis Lectures on Information Security, Privacy, and Trust. Morgan & Claypool Publishers, 2014.

[92] Simson L Garfinkel and Robert C Miller. Johnny 2: A user test of key continuity management with s/mime and outlook express. In *Proceedings of the First Symposium on Usable Privacy and Security*. ACM, jul 2005.

[93] S.L. Garfinkel. Email-based identification and authentication: An alternative to pki? *IEEE Security & Privacy*, 1(6):20–26, nov 2003.

[94] Paolo Gasti and Kasper B. Rasmussen. *Computer Security – ESORICS 2012: 17th European Symposium on Research in Computer Security, Pisa, Italy, September 10-12, 2012. Proceedings*, chapter On the Security of Password Manager Database Formats, pages 770–787. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[95] Shirley Gaw and Edward W Felten. Password management strategies for online accounts. In *Proceedings of the Second Symposium on Usable Privacy and Security*. ACM, jul 2006.

[96] Martin Georgiev, Subodh Iyengar, Suman Jana, Rishita Anubhai, Dan Boneh, and Vitaly Shmatikov. The most dangerous code in the world: Validating ssl certificates in non-browser software. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 38–49, New York, NY, USA, 2012. ACM.

[97] Clint Gibler, Ryan Stevens, Jonathan Crussell, Hao Chen, Hui Zang, and Heesook Choi. Adrob: Examining the landscape and impact of android application plagiarism. In *Proceedings of the 11th international conference on Mobile systems, applications, and services*, MobiSys '13. ACM, 2013.

[98] Google. Android and security. http://googlemobile.blogspot.de/2012/02/android-and-security.html, February 2012.

[99] Google. Android app signing. http://developer.android.com/tools/publishing/app-signing.html, May 2014.

[100] Michael Grace, Yajin Zhou, Qiang Zhang, Shihong Zou, and Xuxian Jiang. RiskRanker: Scalable and Accurate Zero-day Android Malware Detection. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, MobiSys '12. ACM, 2012.

[101] G. Greenwald. *No Place to Hide: Edward Snowden, the NSA and the Surveillance State*. Penguin UK, 2014.

[102] Internet Security Research Group. Let's encrypt. https://letsencrypt.org/.

[103] The Guardian. The nsa files. http://www.theguardian.com/us-news/the-nsa-files.

[104] Saikat Guha, Kevin Tang, and Paul Francis. Noyb: Privacy in online social networks. In *Proceedings of the First Workshop on Online Social Networks*, pages 49–54. ACM, 2008.

[105] Stuart Haber and W. Scott Stornetta. How to time-stamp a digital document. In *Advances in Cryptology - CRYPTO '90*, CRYPTO '90, 1991.

[106] Alina Hang, Alexander De Luca, Matthew Smith, and Michael Richter. Where Have You Been ? Using Location-Based Security Questions for Fallback Authentication. *Symposioum on Usable Privacy and Security*, pages 169–183, 2015.

[107] S M Taiabul Haque, Matthew Wright, and Shannon Scielzo. A study of user password strategy for multiple accounts. In *Proc. CODASPY*. ACM, 2013.

[108] Marian Harbach, Sascha Fahl, and Matthew Smith. Who's Afraid of Which Bad Wolf? A Survey of IT Security Risk Awareness. In *Proc. CSF*, 2014.

[109] Marian Harbach, Markus Hettig, Susanne Weber, and Matthew Smith. Using personal examples to improve risk communication for security and privacy decisions. In *CHI '14 - Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2014.

[110] Boyuan He, Vaibhav Rastogi, Yinzi Cao, Yan Chen, V.N. Venkatakrishnan, Runqing Yang, and Zhenrui Zhang. Vetting ssl usage in applications with sslint. In *Proceedings of the 36th IEEE Symposium on Security and Privacy (Oakland)*, 2015.

[111] C Herley and P Van Oorschot. A research agenda acknowledging the persistence of passwords. *IEEE Security & Privacy*, 10(1):28–36, 2012.

[112] J Hodges, C Jackson, and A Barth. HTTP Strict Transport Security (HSTS). `https://tools.ietf.org/html/rfc6797`, November 2012.

[113] Ralph Holz, Lothar Braun, Nils Kammenhuber, and Georg Carle. The SSL Landscape: A Thorough Analysis of the x.509 PKI Using Active and Passive Measurements. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, IMC '11, 2011.

[114] Philip G Inglesant and M Angela Sasse. The true cost of unusable password policies: password use in the wild. In *Proc. CHI*. ACM, apr 2010.

[115] Family Health International, N. Mack, C. Woodsong, and United States. Agency for International Development. *Qualitative Research Methods: A Data Collector's Field Guide*. FLI, 2005.

[116] C. Jackson and A. Barth. Forcehttps: Protecting high-security web sites from network attacks. In *Proceeding of the 17th International Conference on World Wide Web*, pages 525–534, 2008.

[117] S Jana and V Shmatikov. Memento: Learning Secrets from Process Footprints. *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 143–157, 2012.

[118] M. Just and D. Aspinall. Personal choice and challenge questions: a security and usability assessment. *Proceedings of the 5th Symposium on Usable Privacy and Security*, page 8, 2009.

[119] B. Kaliski. PKCS #5: Password-Based cryptography specification version 2.0, September 2000.

[120] Poul-Henning Kamp. Linkedin password leak: Salt their hide. *Queue*, 10(6):20:20–20:22, June 2012.

[121] Ruogu Kang, Laura Dabbish, Nathaniel Fruchter, and Sara Kiesler. "My data just goes everywhere": User mental models of the internet and implications for privacy and security. *Symposium on Usable Privacy and Security (SOUPS) 2015*, pages 39–52, 2015.

[122] Ambarish Karole, Nitesh Saxena, and Nicolas Christin. A comparative usability evaluation of traditional password managers. In *Proceedings of the 13th International Conference on Information Security and Cryptology*, ICISC'10, pages 233–251, Berlin, Heidelberg, 2011. Springer-Verlag.

[123] Patrick Gage Kelley, Lorrie Faith Cranor, and Norman Sadeh. Privacy as part of the app decision-making process. In *CHI '13 - Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2013.

[124] Patrick Gage Kelley, Saranga Komanduri, Michelle L Mazurek, Rich Shay, Timothy Vidas, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, and Julio Lopez. Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. In *Proc. IEEE S&P*, pages 523–537, 2012.

[125] Hassan Khan, Urs Hengartner, and Daniel Vogel. Usability and Security Perceptions of Implicit Authentication : Convenient , Secure , Sometimes Annoying. *Eleventh Symposium On Usable Privacy and Security (SOUPS 2015)*, pages 225–239, 2015.

[126] Hyoungschick Kim and Rakesh B Bobba. On the Memorability of System-generated PINs : Can Chunking Help ? In *Proceedings of the Eleventh Symposium On Usable Privacy and Security (SOUPS 2015)*, pages 197–209, 2015.

[127] Soo Hyeon Kim, Daewan Han, and Dong Hoon Lee. Predictability of android openssl's pseudo random number generator. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, CCS '13, pages 659–668, New York, NY, USA, 2013. ACM.

[128] Tiffany Hyun-Jin Kim, Lin-Shung Huang, Adrian Perring, Collin Jackson, and Virgil Gligor. Accountable Key Infrastructure (AKI): A Proposal for a Public-key Validation Infrastructure. In *Proceedings of the 2013 Conference on World Wide Web*, WWW '13, 2013.

[129] Saranga Komanduri, Richard Shay, Lorrie Faith Cranor, Cormac Herley, and Stuart Schechter. Telepathwords: Preventing weak passwords by reading users' minds. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 591–606, San Diego, CA, August 2014. USENIX Association.

[130] 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 *Proc. CHI*. ACM, 2011.

[131] Georgios Kontaxis, Elias Athanasopoulos, Georgios Portokalidis, and Angelos D. Keromytis. Sauth: Protecting user accounts from password database leaks. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 187–198, New York, NY, USA, 2013. ACM.

[132] Michael Kranch and Joseph Bonneau. Upgrading HTTPS in mid-air: An empirical study of strict transport security and key pinning. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2014*, 2015.

[133] Alex P Lambert, Stephen M Bezek, and Karrie G Karahalios. Waterhouse: Enabling secure e-mail with social networking. In *Proceedings of the International Conference On Human Factors In Computing Systems*. ACM, April 2009.

[134] B. Laurie and E. Kasper. Revocation transparency, 2013.

[135] B. Laurie, A. Langley, and E. Kasper. Certificate transparency, June 2013.

[136] Ben Laurie, Adam Langley, and Emilia Kasper. Certificate transparency. network working group internet-draft, v12, work in progress, April 2013.

[137] Jonathan Lazar, Jinjuan Heidi Feng, and Harry Hochheiser. *Resarch Methods in Human-Computer Interaction*. Wiley, 2010.

[138] Zhiwei Li, Warren He, Devdatta Akhawe, and Dawn Song. The emperor's new password manager: Security analysis of web-based password managers. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 465–479, San Diego, CA, August 2014. USENIX Association.

[139] libsodium. The sodium crypto library (libsodium). `https://download.libsodium.org/doc/`.

[140] Matthew M. Lucas and Nikita Borisov. Flybynight: Mitigating the privacy risks of social networking. In *Proceedings of the 7th ACM Workshop on Privacy in the Electronic Society*, pages 1–8, 2008.

[141] David Malone and Kevin Maher. Investigating the distribution of password choices. In *Proceedings of the 21st international conference on World Wide Web*, WWW '12, pages 301–310, New York, NY, USA, 2012. ACM.

[142] M. Marlinspike. More tricks for defeating ssl in practice. In *Black Hat USA*, 2009.

[143] M. Marlinspike. New Tricks for Defeating SSL in Practice. In *Black Hat Europe*, 2009.

[144] Moxie Marlinspike. Convergence. `http://convergence.io/`.

[145] Moxie Marlinspike. Tack: Trust assertion for certificate keys. http://tack.io/.

[146] Moxie Marlinspike. Ssl and the future of authenticity. In *BlackHat USA 2011*, 2011.

[147] Vitaly Shmatikov Martin Georgiev, Suman Jana. Breaking and fixing origin-based access control in hybrid web/mobile application frameworks. In *NDSS 2014: 21st Network & Distributed System Security Symposium*. Internet Society, 2014.

[148] Max-Emanuel Maurer, Alexander De Luca, and Sylvia Kempe. Using data type based security alert dialogs to raise online security awareness. In *Proceedings of the Seventh Symposium on Usable Privacy and Security*. ACM, 2011.

[149] 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.

[150] David McCandless. World's biggest data breaches. http://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/.

[151] P McDaniel and William Enck. Not So Great Expectations: Why Application Markets Haven't Failed Security. *IEEE Security & Privacy*, 8(5):76–78, September 2010.

[152] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1996.

[153] Mohammad Nauman, Sohail Khan, and Xinwen Zhang. Apex: Extending android permission model and enforcement with user-defined runtime constraints. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, 2010.

[154] National Institute of Standards and Technology (NIST). Advanced encryption standard (aes) (fips pub 197), October 2001.

[155] Marten Oltrogge, Yasemin Acar, Sergej Dechand, Matthew Smith, and Sascha Fahl. To pin or not to pin—helping app developers bullet proof their tls connections. In *Proceedings of the 24th USENIX Security Symposium*. USENIX, 2015.

[156] R Pandita, X Xiao, W Yang, W Enck, and T Xie. WHYPER: towards automating risk assessment of mobile applications. In *Proceedings of the 22nd USENIX Security Symposium*, 2013.

[157] Thanasis Petsas, Antonis Papadogiannakis, Michalis Polychronakis, Evangelos P. Markatos, and Thomas Karagiannis. Rise of the planet of the apps: A systematic study of the mobile app ecosystem. In *Proceedings of the 2013 ACM SIGCOMM conference on Internet measurement conference*, IMC '13, 2013.

[158] Sebastian Poeplau, Yanick Fratantonio, Antonio Bianchi, Christopher Kruegel, and Giovanni Vigna. Execute this! analyzing unsafe and malicious dynamic code loading in android applications. In *NDSS 2014: 21st Network & Distributed System Security Symposium*. Internet Society, 2014.

[159] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. Android permissions demystified. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*. ACM Press, oct 2011.

[160] Adrienne Porter Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner. Android Permissions: User Attention, Comprehension, and Behavior. Technical report, UC Berkeley, 2012.

[161] Adrienne Porter Felt, Helen J. Wang, Alexander Moshchuk, Steve Hanna, and Erika Chin. Permission re-delegation: Attacks and defenses. In *Proceedings of the 20th USENIX Security Symposium*. USENIX, 2011.

[162] Georgios Portokalidis, Philip Homburg, Kostas Anagnostakis, and Herbert Bos. Paranoid android: Versatile protection for smartphones. In *Proceedings of the 26th Annual Computer Security Applications Conference*, dec 2010.

[163] Hootan Rashtian, Yazan Boshmaf, Pooya Jaferian, and Konstantin Beznosov. To Befriend Or Not? A Model of Friend Request Acceptance on Facebook. *SOUPS '14: Proceedings of the Tenth Symposium On Usable Privacy and Security*, pages 285–300, 2014.

[164] Vaibhav Rastogi, Yan Chen, and Xuxian Jiang. DroidChameleon: evaluating Android anti-malware against transformation attacks. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, ASIA CCS '13. ACM, 2013.

[165] Rob Reeder and Sunny Consolvo. "... no one can hack my mind": Comparing Expert and Non-Expert Security Practices. *Symposium on Usable Privacy and Security*, pages 327–346, 2015.

[166] E. Rescorla. HTTP Over TLS. RFC 2818, May 2000.

[167] Phillip Rogaway and David Wagner. Comments to nist concerning aes modes of operations: Ctr-mode encryption. National Institute of Standards and Technologies, 2000.

[168] Blake Ross, Collin Jackson, Nick Miyake, Dan Boneh, and John C. Mitchell. Stronger password authentication using browser extensions. In *Proceedings of the 14th Conference on USENIX Security Symposium - Volume 14*, SSYM'05, pages 2–2, Berkeley, CA, USA, 2005. USENIX Association.

[169] Scott Ruoti, Nathan Kim, Ben Burgon, Timothy van der Horst, and Kent Seamons. Confused johnny: When automatic encryption leads to confusion and mistakes. In *Proceedings of the Ninth Symposium on Usable Privacy and Security*, SOUPS '13, pages 5:1–5:12, New York, NY, USA, 2013. ACM.

[170] Mark D. Ryan. Enhanced certificate transparency and end-to-end encrypted mail. In *NDSS 2014: 21st Network & Distributed System Security Symposium*, NDSS 2014, 2014.

[171] P. Saint-Andre and J. Hodges. Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS), mar 2011.

[172] Stuart Schechter and Joseph Bonneau. Learning Assigned Secrets for Unlocking Mobile Devices. In *Proceedings of the Eleventh Symposium On Usable Privacy and Security (SOUPS 2015)*, pages 277–295, 2015.

[173] Stuart E. Schechter, Rachna Dhamija, Andy Ozment, and Ian Fischer. The emperor's new security indicators. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, SP '07, pages 51–65, Washington, DC, USA, 2007. IEEE Computer Society.

[174] R Schlegel, K Zhang, X Zhou, M Intwala, and et al. Soundcomber: A stealthy and context-aware sound trojan for smartphones. *Proceedings of the Network and Distributed System Security Symposium*, 2011.

[175] A Shabtai, Y Fledel, U Kanonov, Y Elovici, S Dolev, and C Glezer. Google android: A comprehensive security assessment. *Security & Privacy, IEEE*, 8(2):35–44, 2010.

[176] 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*, SOUPS '12, pages 7:1–7:20, New York, NY, USA, 2012. ACM.

[177] 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 SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 2927–2936, New York, NY, USA, 2014. ACM.

[178] 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.

[179] Steve Sheng, Colleen Koranda, Jeremy Hyland, and Levi Broderick. Why Johnny Still Can't Encrypt: Evaluating the Usability of Email Encryption Software. In *Proceedings of the Second Symposium on Usable Privacy and Security, Poster*, 2006.

[180] D. Shin and R. Lopes. An empirical study of visual security cues to prevent the sslstripping attack. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pages 287–296, 2011.

[181] David Silver, Suman Jana, Dan Boneh, Eric Chen, and Collin Jackson. Password managers: Attacks and defenses. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 449–464, San Diego, CA, August 2014. USENIX Association.

[182] Yimin Song, Chao Yang, and Guofei Gu. Who is peeping at your passwords at starbucks? – to catch an evil twin access point. In *IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 323–332, 2010.

[183] A Sotirakopoulos and K Hawkey. "i did it because i trusted you": Challenges with the study environment biasing participant behaviours. In *Proceedings of the 6th Symposium on Usable Privacy and Security*, 2010.

[184] Andreas Sotirakopoulos, Kirstie Hawkey, and Konstantin Beznosov. On the challenges in usable security lab studies: Lessons learned from replicating a study on ssl warnings. In *Proceedings of the 7th Symposium on Usable Privacy and Security*, jul 2011.

[185] David Sounthiraraj, Justin Sahs, Garrett Greenwood, Zhiqiang Lin, and Latifur Khan. Smv-hunter: Large scale, automated detection of ssl/tls man-in-the-middle vulnerabilities in android apps. In *NDSS 2014: 21st Network & Distributed System Security Symposium.* Internet Society, 2014.

[186] Elizabeth Stobert and Robert Biddle. The password life cycle: User behaviour in managing passwords. In *Proceedings of the Tenth Symposium on Usable Privacy and Security*, pages 243–255, Menlo Park, CA, July 2014. USENIX Association.

[187] Ben Stock and Martin Johns. Protecting users against xss-based password manager abuse. In *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '14, pages 183–194, New York, NY, USA, 2014. ACM.

[188] Joshua Sunshine, Serge Egelman, Hazim Almuhimedi, Neha Atri, and Lorrie Faith Cranor. Crying wolf: An empirical study of ssl warning effectiveness. In *Proceedings of the 18th Conference on USENIX Security Symposium*, SSYM'09, pages 399–416, Berkeley, CA, USA, 2009. USENIX Association.

[189] Xiaoyuan Suo, Ying Zhu, and G. Scott. Owen. Graphical passwords: A survey. In *Proceedings of the 21st Annual Computer Security Applications Conference*, ACSAC '05, pages 463–472, Washington, DC, USA, 2005. IEEE Computer Society.

[190] Julie Thorpe, Brent MacRae, and Amirali Salehi-Abari. Usability and security evaluation of geopass: A geographic location-password scheme. In *Proceedings of the Ninth Symposium on Usable Privacy and Security*, SOUPS '13, pages 14:1–14:14, New York, NY, USA, 2013. ACM.

[191] Sean Turner and Blake C. Ramsdell. Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification. IETF RFC 5751, January 2010.

[192] Nik Unger, Sergej Dechand, Joseph Bonneau, Sascha Fahl, Henning Perl, Ian Goldberg, and Matthew Smith. Sok: Secure messaging. In *IEEE Symposium on Security and Privacy*, pages 232–249. IEEE Computer Society, May 2015.

[193] Blase Ur, Patrick Gage Kelley, Saranga Komanduri, Joel Lee, Michael Maass, Michelle L. Mazurek, Timothy Passaro, Richard Shay, Timothy Vidas, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. How does your password measure up? the effect of strength meters on password creation. In *Proceedings of the 21st USENIX conference on Security symposium*, Security'12, pages 5–5, Berkeley, CA, USA, 2012. USENIX Association.

[194] 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 *Proceedings of the Eleventh Symposium On Usable Privacy and Security (SOUPS 2015)*. USENIX, July 2015.

[195] Vasco.com. http://www.vasco.com/company/about_vasco/press_room/news_archive/2011/news_diginotar_reports_security_incident.aspx, September 2011.

[196] T. Vidas, D. Votipka, and N. Christin. All your droid are belong to us: A survey of current android attacks. In *Proceedings of the 5th USENIX conference on Offensive technologies*, pages 10–10, 2011.

[197] Timothy Vidas and Nicolas Christin. Sweetening android lemon markets: Measuring and combating malware in application marketplaces. In *CODASPY 13: Proceedings of the Third ACM Conference on Data and Application Security and Privacy*, CODASPY '13, 2013.

[198] M. Weir, S. Aggarwal, B. de Medeiros, and B. Glodek. Password cracking using probabilistic context-free grammars. In *Proc. IEEE S&P*, pages 391–405, 2009.

[199] Matt Weir, Sudhir Aggarwal, Michael Collins, and Henry Stern. Testing metrics for password creation policies by attacking large sets of revealed passwords. In *Proc. ACM CCS*, pages 162–175, 2010.

[200] Dan Wendlandt, David G. Andersen, and Adrian Perrig. Perspectives: improving SSH-style host authentication with multi-path probing. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, ATC'08, pages 321–334, Berkeley, CA, USA, 2008. USENIX Association.

[201] A. Whitten and J.D. Tygar. Why johnny can't encrypt: A usability evaluation of pgp 5.0. In *Proceedings of the 8th USENIX Security Symposium*, 1999.

[202] Lei Wu, Michael Grace, Yajin Zhou, Chiachih Wu, and Xuxian Jiang. The impact of vendor customizations on android security. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & Communications Security*, CCS '13, 2013.

[203] T Wu. The SRP Authentication and Key Exchange System. https://www.ietf.org/rfc/rfc2945.txt, September 2000.

[204] Lok Kwong Yan and Heng Yin. Droidscope: Seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis. In *Proceedings of the 21st USENIX Security Symposium*, USENIX Security'12, 2012.

[205] Y Ylonen. The Secure Shell (SSH) Transport Layer Protocol. https://www.ietf.org/rfc/rfc4253.txt, January 2006.

[206] Nur Haryani Zakaria, David Griffiths, Sacha Brostoff, and Jeff Yan. Shoulder surfing defence for recall-based graphical passwords. In *Proceedings of the Seventh Symposium on Usable Privacy and Security*, page 6. ACM, 2011.

[207] Wu Zhou, Xinwen Zhang, and Xuxian Jiang. AppInk: Watermarking Android Apps for Repackaging Deterrence. In *ASIA CCS '13: Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, ASIA CCS '13. ACM, 2013.

[208] Wu Zhou, Yajin Zhou, Xuxian Jiang, and Peng Ning. Detecting repackaged smartphone applications in third-party android marketplaces. In *Proceedings of the second ACM conference on Data and Application Security and Privacy*. ACM, 2012.

[209] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang. Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets. In *Proceedings of the 19th Annual Network and Distributed System Security Symposium*, 2012.

[210] Yajin Zhou and Xuxian Jiang. Dissecting android malware: Characterization and evolution. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 95–109. IEEE, 2012.

[211] Philip R. Zimmermann. *The Official PGP User's Guide*. MIT Press, Cambridge, MA, USA, 1995.

Curriculum Vitae

CURRICULUM VITAE: Sascha Fahl

Personal Information

Name	Sascha Fahl
Date of birth	23.03.1985
City of birth	Eschwege, Germany

Education

since 02/2011	**PhD Computer Science** Leibniz University Hannover, Germany. Distributed Computing and Security Group. Areas of research: Human Factors in IT Security and Privacy for End Users, Administrators, Software Developers and System Designers, Android Security, Transport Layer Security, Transparency Systems.
10/2005 – 01/2011	**Diplom Informatik** University of Marburg, Germany. Topic of thesis: "*Identity Based Encryption for Cloud Computing*"
08/2001 – 06/2004	**Abitur (A-levels)** Berufliches Gymnasium, Eschwege, Germany (high school with professional specialization). Specialization in Economics.

Experience

PC Member at: SOUPS 2016, TRUST 2015, ACM WWW 2015, Usable Security Workshop 2014 co-located with NDSS, TRUST 2014, IEEE DEST 2012, CALS 2011. Reviewer for: IEEE Security and Privacy 2015, ACM CHI 2015, ACM CCS 2015, SOUPS 2015, SOUPS 2014, ACM CCS 2014, IEEE Security and Privacy 2014 and Financial Crypto 2013. Subreviewer for: CHI 2014.

CISPA, Saarland University, Germany (Oct. 2015 - now): Researcher working in the field of Usable Security and Privacy; Teaching Seminars on Usable Security and Privacy; Advising three PhD students; Advising 4 students on Bachelor and Master theses.

Leibniz University Hannover, Germany (Feb. 2011 - Sep. 2015): Teaching assistant on IT security and Usable Security and Privacy courses; Seminars on Security and Privacy in Society (interdisciplinary course in conjunction with the School of Sociology and the School of Law) and introductory seminars on IT Security; Supervising students in Security and Usable Security project work; Advising 18 students on Bachelor and Master theses.

FKIE Fraunhofer, Bonn (Jan. 2014 - Feb. 2016): Working on multiple usable security projects as part-time research assistant.

Interned with the Google Chrome Usable Security Team, Mountain View, California (Spring 2015): Big Data Analysis and Designing and Implementing a new TLS interstitial for Chrome.

Peer-reviewed Papers (selected)

Y. Acar, M. Backes, S. Bugiel, S. Fahl, P. McDaniel, M. Smith: SoK: Lessons Learned From Android Security Research For Appified Software Platforms, IEEE Security and Privacy 2016

Y. Acar, M. Backes, S. Fahl, D. Kim, M. Mazurek, C. Stransky: You Get Where You're Looking For - The Impact of Information Sources on Code Security, IEEE Security and Privacy 2016

H. Perl, D. Arp, S. Dechand, S. Fahl, Y. Acar, F. Yamaguchi, K. Rieck, M. Smith: VCCFinder: Finding Potential Vulnerabilities in Open-Source Projects to Assist Code Audits, ACM CCS 2015

M. Oltrogge, Y. Acar, S. Dechand, M. Smith, S. Fahl: To Pin or Not to Pin - Helping App Developers To Bulletproof Their TLS Connections, USENIX Security 2015

N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, M. Smith: SoK: Secure Messaging, IEEE Security and Privacy 2015

S. Fahl, Y. Acar, H. Perl, M. Smith: Why Eve and Mallory (Also) Love Webmasters: A Study on the Root Causes of SSL Misconfigurations, ACM AsiaCCS 2014

S. Fahl, S. Dechand, H. Perl, F. Fischer, J. Smrceck, M. Smith: Hey, NSA: Stay Away from my Market! Future Proofing App Markets against Powerful Attackers, ACM CCS 2014

S. Fahl, M. Harbach, H. Perl, M. Koetter, M. Smith: Rethinking SSL Development in an Appified World, ACM CCS 2013

S. Fahl, M. Harbach, Y. Acar, M. Smith: On the Ecological Validity of a Password Study, Symposium on Usable Privacy and Security, SOUPS 2013

S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, M. Smith: Why Eve and Mallory Love Android: An Analysis of Android SSL (In)Security, ACM CCS 2012

S. Fahl, M. Harbach, T. Muders, M. Smith, U. Sander: Helping Johnny 2.0 to Encrypt his Facebook Conversations, Symposium on Usable Privacy and Security, SOUPS 2012