

An Analysis of Password Manager Security and Usage on Desktop and Mobile Devices

A Dissertation Presented for the
Doctor of Philosophy
Degree

The University of Tennessee, Knoxville

Timothy Sean Oesch

May 2021

© by Timothy Sean Oesch, 2021
All Rights Reserved.

I would like to dedicate this work to my family, whose unwavering love and support have been a constant source of encouragement throughout the PhD process, and to all those committed to defending the world against the dark arts.

Acknowledgments

I would like to thank my advisor, Dr. Scott Ruoti, without whom this work would not have been possible, and my dissertation committee (Dr. Kent Seamons, Dr. Jinyuan Sun, and Dr. Doowon Kim) who provided timely feedback and direction. I would also like to thank my fellow students James Simmons and Bhaskar Gaur for their feedback and support, and especially Anuj Gautam, who provided invaluable support on two of my papers. I would also like to thank my coworkers who participated in pilot studies and encouraged me throughout the process of completing my studies. And last but certainly not least, I extend a heartfelt thanks to my family, who endured the long hours of studying and were always there to encourage me during this journey.

Abstract

Security experts recommend password managers to help users generate, store, and enter strong, unique passwords. Prior research confirms that managers do help users move towards these objectives, but it also identified usability and security issues that had the potential to leak user data or prevent users from making full use of their manager. In this dissertation, I set out to measure to what extent modern managers have addressed these security issues on both desktop and mobile environments. Additionally, I have interviewed individuals to understand their password management behavior.

I begin my analysis by conducting the first security evaluation of the full password manager lifecycle (generation, storage, and autofill) on desktop devices, including the creation and analysis of a corpus of 147 million generated passwords. My results show that a small percentage of generated passwords are weak against both online and offline attacks, and that attacks against autofill mechanisms are still possible in modern managers. Next, I present a comparative analysis of autofill frameworks on iOS and Android. I find that these frameworks fail to properly verify webpage security and identify a new class of phishing attacks enabled by incorrect handling of autofill within WebView controls hosted in apps. Finally, I interview users of third-party password managers to understand both how and why they use their managers as they do. I find evidence that many users leverage multiple password managers to address issues with existing managers, as well as provide explanations for why password reuse continues even in the presence of a password manager. Based on these results, I conclude with recommendations addressing the attacks and usability issues identified in this work.

Table of Contents

1	Introduction	1
2	Background	5
2.1	Password Managers	5
2.2	Related Work	6
2.2.1	Desktop Managers	7
2.2.2	Mobile Autofill	8
2.2.3	Usability and User Studies	10
2.2.4	Relation to This Work	12
3	Browser-Based Password Managers on the Desktop	14
3.1	Analyzed Password Managers	16
3.1.1	App	16
3.1.2	Extension	18
3.1.3	Browser	19
3.1.4	Updates for Password Managers	20
3.2	Password Generation	20
3.2.1	Settings and Features	20
3.2.2	Password Collection and Analysis	22
3.2.3	Results	26
3.3	Password Storage	34
3.3.1	Password Vault Encryption	34
3.3.2	Metadata Privacy	36

3.4	Password Autofill	37
3.4.1	User Interaction Requirements	37
3.4.2	Autofill for iframes	40
3.4.3	Fill Form Differing from Saved Form	41
3.4.4	Non-Standard Login Fields	42
3.4.5	Potential Mitigation	42
3.4.6	Web Vault Security & Bookmarklets	43
3.5	Discussion	44
3.5.1	Filter weak passwords.	46
3.5.2	Better master password policies.	46
3.5.3	Safer autofill.	46
4	A Security Analysis of Autofill Frameworks on iOS and Android	48
4.1	Background	50
4.1.1	Password Managers	50
4.1.2	Secure Autofill	51
4.1.3	WebView	52
4.2	Evaluation Methodology	53
4.2.1	Mobile Autofill Frameworks	54
4.2.2	Testing Approach	56
4.3	Browser Autofill	58
4.3.1	P1—User Interaction	60
4.3.2	P2—Credential-to-Domain Mapping	62
4.3.3	P3—Protecting Filled Credentials	62
4.4	App Autofill	64
4.4.1	P1—User Interaction	64
4.4.2	P2—Credential-to-App Mapping	67
4.4.3	P3—Protecting Filled Credentials	69
4.5	WebView Autofill	70
4.5.1	P1—User Interaction	73

4.5.2	P2—Credential-to-Domain Mapping	73
4.5.3	P3—Protecting Filled Credentials	75
4.6	Password Generation	76
4.6.1	Comparison to Desktop Managers	79
4.7	Password Storage	79
4.7.1	Password Encryption	82
4.7.2	Metadata Privacy	83
4.7.3	Comparison to Desktop Managers	83
4.8	Discussion	83
4.8.1	Addressing WebView Phishing Attacks	86
4.8.2	Recommendations for Secure Autofill Frameworks	87
4.8.3	Autofill Framework Smells	89
4.8.4	Future Research	89
5	"It basically started using me" - An Observational Study of Password Manager Usage	92
5.1	Methodology	95
5.1.1	Recruitment	95
5.1.2	Interview Design	96
5.1.3	Addressing Potential Risks	99
5.1.4	Analysis	99
5.1.5	Participant Demographics	100
5.1.6	Limitations	101
5.2	Core Theories	101
5.2.1	Adoption of multiple managers	101
5.2.2	Cross-device entry and reuse	104
5.2.3	Limited usage on mobile	109
5.2.4	Breach notifications desired, but health check overwhelming	111
5.2.5	Adoption and promotion	115
5.3	Additional Topics	116

5.3.1	Password sharing via manager difficult and rarely used	116
5.3.2	Mental models of password reuse	119
5.3.3	Inconsistencies in autofill and autosave	120
5.3.4	Desire for single sign-on	122
5.3.5	Ubiquitous usage of default settings	122
5.3.6	Attitudes towards autolock	123
5.3.7	How COVID-19 changed password manager usage	123
5.4	Discussion	124
5.4.1	Reducing Multiple Manager Usage.	125
5.4.2	Addressing Concerns with Generated Passwords	126
5.4.3	Health Check Should Be Seamless and Targeted.	127
6	Conclusion and Future Work	128
6.1	Lessons Learned	129
6.1.1	Managers need better support.	129
6.1.2	Inconsistencies lead to reuse.	130
6.1.3	Usability issues prevent full adoption.	130
6.1.4	User concerns prevent full adoption.	130
6.1.5	Inaccurate mental models are still an issue.	131
6.1.6	Secure autofill should be a priority.	131
6.2	Future Work	131
6.2.1	Browser-Supported Password Managers.	131
6.2.2	Helping Users with Cross-device Entry	132
6.2.3	Secure Credential Mapping.	133
6.2.4	Research-Derived Character Sets.	133
6.2.5	Autofill Framework for the Desktop.	134
	Bibliography	135
	Appendices	145
A	Evolution of Mobile Autofill	146

A.1	Autofill on iOS	146
A.2	Autofill on Android	148
B	Password Manager Download Statistics	150
C	Password Manager Browser Evaluation Results	152
D	Android Credential Mapping Details	155
E	Observational Study Instruments	157
E.1	Screening Survey	157
E.2	Interview Sign Up Survey	159
E.3	Semi-structured Interview Script	160
E.4	Consent Form for Interview	167
E.5	Consent Form for Screening Survey	169
Vita		172

List of Tables

3.1	Analyzed Password Managers	17
3.2	Overview of Password Generation Features	21
3.3	Character Frequencies for length 20 passwords using all characters. Groups of similar characters represent a requirement to include at least one character from that set, causing characters from smaller sets to be selected with greater frequency.	28
3.5	Length 8 χ^2 Scores for Character Frequency	29
3.6	Length 12 χ^2 Scores for Character Frequency	30
3.7	Length 20 χ^2 Scores for Character Frequency	31
3.4	χ^2 test for random character distribution	33
3.8	Randomly Generated Weak Passwords	33
3.9	Overview of Password Vault Encryption	35
3.10	Overview of Password Autofill Features	38
4.1	Analyzed password managers on iOS and Android	59
4.2	Autofill Security in Mobile Browsers	61
4.3	Autofill Security for Native UI Elements in Apps	65
4.4	App-to-Domain Mappings on Android	71
4.5	Autofill Security for WebView Controls	72
4.6	Password Generation Features	81
4.7	Overview of Password Vault Encryption	84
4.8	Overview of Password Vault Metadata Behavior	85
1	Autofill in mobile browsers on iOS	153

2	Autofill in mobile browsers on Android	154
---	--	-----

List of Figures

3.1	Neural Network Guess Estimates ($\log_{10}$). Differences are primarily attributed to character set size.	27
4.1	iOS App Extensions UI	55
4.2	iOS Password AutoFill UI	57
4.3	Android Password AutoFill UI	65
4.4	Example of a cross-origin iframe phishing attack	66
4.6	Examples of password generation configuration.	80
4.7	Warnings displayed by Dashlane and Smart Lock when a secure mapping could not be established	91
5.1	Concept map for multiple manager usage.	105
5.2	Concept map for password reuse and its causes	107
5.3	Concept map for when managers stop reuse.	108
5.4	Concept map for limited manager usage on mobile devices.	112
5.5	Concept map for account auditing and health check.	114
5.6	Concept map for adoption.	117
5.7	Concept map for promotion.	118
1	Google Play Store Downloads from March 2020	150
2	iOS Download Estimates from April 2020	151

Chapter 1

Introduction

Despite the well-established problems facing password-based authentication, it continues to be the dominant form of authentication used on the web [11]. Because passwords that are difficult for an attacker to guess are also hard for users to remember, users often create weaker passwords to avoid the cognitive burden of recalling them [18, 53]. In fact, with the increase in the number of passwords users are required to store, they often reuse passwords across websites [17, 26, 50, 66]. Herley points out that this rejection of security advice by users is rational when the low percentage of users affected by breaches is contrasted with the effort required [35]. However, the number of data breaches is on the rise [55], and this situation leaves many users vulnerable to exploitation.

Password managers provide a mechanism for users to create, store, and fill strong, unique passwords. Because managers remember users passwords for them, users can generate strong passwords without the fear of forgetting or losing them. Most managers provide a generation pop-up dialogue next to password fields to encourage users to create strong passwords and periodically remind users if they still have weak or reused passwords. Moreover, research confirms that managers do reduce password reuse among their users [41].

However, prior work identifies security challenges present in managers. The password autofill functionality may allow attackers to steal or phish users' credentials [39, 58, 60]. Additionally, password vaults occasionally leave users' credentials or data vulnerable to attack [29, 21], including unencrypted metadata and side channel leakage from encrypted information. No prior work examines the security of password generation in managers.

Because no prior work has analyzed the entire password lifecycle (generation, storage, autofill) in password managers, I begin my dissertation by examining the security of the full lifecycle in desktop managers. I find that autofill is commonly vulnerable to XSS based attacks, which in some cases allow an attacker to scrape all of a user’s credentials at once. The autofill related security vulnerabilities I identify on desktop raise questions about the state of autofill security on mobile, where there is OS support for autofill that is not available on the desktop and where autofill occurs in a wider variety of contexts. I continue my dissertation by evaluating the security of these mobile autofill frameworks. What I find is that mobile autofill frameworks overall are less secure than their desktop counterparts, leaving users vulnerable to phishing attacks inside WebView and failing to provide a secure mapping between websites and apps. In addition to these concerns, there are also many inconsistencies in behavior that impact usability. To better understand the usability issues facing password management generally, I conduct a cross-device observational interview of password manager users. My analysis of these interviews, which utilizes grounded theory, identifies five key processes describing manager usage, including why and how users often use multiple managers simultaneously and a correlation between problems with cross-device password entry and password reuse among manager users. Overall, my dissertation reveals a need for browsers and OSes to provide additional support for password management, as well as for managers to address usability concerns that lead to password reuse.

In Chapter 3, I provide details about my security evaluation of the entire password lifecycle (generation, storage, and autofill) in desktop managers. For password generation, I evaluate a corpus of 147 million passwords generated by the studied password managers to determine whether they exhibit any non-randomness that an attacker could leverage. My results find several issues with the generated passwords, the most severe being that a small percentage of shorter generated passwords are weak against online and offline attacks (shorter than 10 characters and 18 characters, respectively). I also replicate earlier work examining the security of password storage [29] and autofill [39, 58, 60]. While password managers have improved in the past five years, there are still significant security concerns, especially regarding autofill in the browser.

In contrast to desktop environments, mobile operating systems provide autofill frameworks that attempt to provide a more unified and secure autofill experience. I continue my dissertation by evaluating the security of these autofill mechanisms as well as password manager security on mobile generally (see Chapter 4). This evaluation covers three mobile autofill frameworks, each of which takes a different approach to how much of the autofill process is handled by the framework: iOS’s app extensions (mostly handled by the password manager), iOS’s Password AutoFill (mostly handled by the framework), and Android’s autofill service (somewhere in the middle).

My results show that while autofill frameworks can be effective in ensuring correct password manager behavior, existing frameworks are poorly implemented and can be more detrimental than helpful when it comes to security. For mobile browsers, the frameworks fail to properly check the security of the webpage where credentials are to be entered and do not allow password managers to examine the webpage to make these checks themselves (like they do on desktop). For WebView controls, I identify two novel phishing attacks that use the password manager as a confused deputy to aid the attacker in using a malicious app to phish credentials for arbitrary domains or to use a malicious webpage to phish credentials for a legitimate app. Based on these results, I provide concrete recommendations for the design and implementation of a more secure autofill framework.

After exploring the security of managers on both desktop and mobile devices, I turned my attention to understanding how and why users use managers across their devices (see Chapter 5). To this end, I conducted a semi-structured observational interview of third-party password manager (e.g. LastPass, 1Password, Dashlane) users to understand how users utilize managers on their own devices. I found that the persistent availability of browser managers, along with concerns related to losing passwords and cross-device syncing issues, lead users to adopt multiple managers, a phenomenon not reported in prior research. I also found that concerns related to cross-device entry and the fear of non-availability prevented users from using managers for the entire password lifecycle, which directly leads to password reuse, helping explain previous findings [41, 72] that reuse is still common among manager users.

At a high level, this dissertation begins by assessing how managers approach security and then explores how users approach managers. By better understanding both managers and their users, I lay the groundwork for future password managers to be both more usable and more secure. In Chapter 6 I close with a discussion of key lessons learned while conducting this research and suggest directions for future research.

Chapter 2

Background

In this section, I describe the responsibilities of a password manager. I also describe prior work that has analyzed password managers.

2.1 Password Managers

In the most basic sense, a password manager is a tool that stores a user’s credentials (i.e., username and password) to alleviate the cognitive burden associated with a user remembering many unique login credentials [39]. This store of passwords is commonly referred to as a *password vault*. The vault itself is ideally stored in encrypted form, with the encryption key most commonly derived from a user-chosen password known as the *master password*. Optionally, the password vault can be stored online, allowing it to be synchronized across multiple devices.

In addition to storing user-selected passwords, most modern password managers can help users generate passwords. Password generation takes as input the length of the desired password, the desired character set, and any special attribute the password should exhibit (e.g., at least one digit and one symbol, no hard to recognize characters). The password generator outputs a randomly generated password that meets the input criterion.

Many password managers also help users authenticate to websites by automatically selecting and filling in (i.e., *autofill*) the appropriate username and password. If users have

multiple accounts on the website, the password manager will allow users to select which account they wish to use for autofill.

Taken together, password generation, storage, and autofill make up the password manager life cycle [15]:

1. A user navigates to a website to create a new account. The password manager helps the user *generates* a random password that is unique to the current website.
2. The password manager then stores the user’s username and generated password.
3. When the user next visits the website, the password manager autofills the user’s username and generated password.

If properly implemented and used, a password manager has several tangible benefits to the user:

1. It reduces the cognitive burden of remembering usernames and passwords.
2. It is easy to assign a different password to every website, addressing the problem of password reuse.
3. It is easy to generate passwords that are resilient to online *and* offline guessing attacks.

For password managers that store user data on their servers, such as LastPass, the data is always encrypted with the MP prior to being uploaded [41]. To preserve privacy, the MP is not known to the company storing the data and encryption / decryption must occur on the device. In browser-based password managers that allow syncing, such as Chrome, the passwords are protected by the same credentials as the user account on the device.

2.2 Related Work

I discuss the related work in three sections, each relating to a chapter of this dissertation. First, I discuss related work on desktop managers. Second, I discuss work relevant to autofill on mobile devices. Third, I discuss usability and user studies of managers on both desktop and mobile devices. I conclude by discussing how this work fits into the larger context of prior work.

2.2.1 Desktop Managers

While prior work examines autofill and storage security, no prior work examines the security of password generation in managers.

Desktop Autofill Security

Silver et al. [58] studied the autofill feature of ten password managers. They demonstrated that if a password manager autofilled passwords without requiring user interaction, it was possible to steal a user’s credentials for all websites that were vulnerable to a network injection attack or had an XSS vulnerability on any page of the website. They also showed that even if user interaction was required, if autofill was allowed inside an iframe, then the attacker could leverage clickjacking to achieve user interaction without users realizing they were approving the release of their credentials. Stock and Johns [60] also studied autofill related vulnerabilities in six browser-based password managers and had similar findings to Silver et al. Li et al. [39] studied five extension-based password managers and found logic and authorization errors, misunderstandings about the web security model, and CSRF/XSS attacks.

Storage Security

Gasti and Rasmussen [29] analyzed the security of the password vaults used by thirteen password managers, finding a range of vulnerabilities that could leak sensitive information to both passive and active attackers. These vulnerabilities were related to unencrypted metadata as well as side channel information leakage from encrypted data. In addition, a recent study by Independent Security Evaluators [21] found that password managers were not encrypting passwords that they wrote to memory, making it trivial to extract some passwords from the password vault even when it was not in use. Chatterjee et al. [13] and Bojinov et al. [9] proposed alternative password vault schemes that are more resilient to offline attacks, but password managers have not adopted these schemes.

2.2.2 Mobile Autofill

Work discussing phishing attacks and WebView vulnerabilities is relevant to my work on mobile autofill because I identified a novel class of phishing attacks that utilize mobile WebView. Attacks that utilize the accessibility service provided by the mobile OS or the clipboard are also common in research on mobile security, so I include them here for reference.

Mobile Autofill Security

Aonzo et al. [3] demonstrated that malicious apps could trick the Android autofill service into suggesting to the user that they autofill credentials for a legitimate app into the malicious app. These attacks take advantage of poor mapping strategies between apps and online domains used by password managers. They found that Keeper, LastPass, 1Password, and Dashlane were vulnerable to mapping-based attacks, but did not identify any attacks against Google Smart Lock. To address the problems they found, they recommend that all password managers adopt Google Smart Lock’s mapping scheme which relies on Digital Asset Link (DAL) files published to websites that identify which apps should receive credentials for that website.

Like Aonzo et al., I too investigate app-to-domain mappings for Android password managers, expanding Aonzo et al.’s work by examining additional password managers (13 password managers versus the five in Aonzo et al.’s work), finding that app-to-domain mapping issues are widespread on Android. I also conduct the first analysis of app-to-domain mappings on iOS, finding that iOS app extensions takes the most insecure approach to app-to-domain mapping (apps choose what they are mapped to), but that the iOS Password AutoFill framework enforces a secure app-to-domain credential mapping for all password managers. As part of my investigation, I also identify two novel phishing attack that use the WebView widget to sidestep app-to-domain mappings, even those of iOS Password AutoFill.

Feasibility of Phishing Attacks

Phishing attacks have been shown to be effective even against the most sophisticated users when visual deception, such as website redressing or overlays, is used [19]. Felt and Wagner [24]

also found that on mobile devices users are often asked by legitimate apps and websites to autofill credentials after clicking a link. As a result, users become conditioned to providing their credentials after clicking a link, which makes phishing attacks even easier. Luo et al. [40] also demonstrated several effective phishing attacks against WebView on iOS that utilized UI redressing and overlays to steal credentials. Most recently, Tuncay et al. [64] demonstrated that naming policies for Android apps allowed malicious apps to effectively masquerade as legitimate apps when requesting permissions from the user.

Several works have explored ways of protecting users from malicious apps that mimic the GUI of legitimate apps in an attempt perform phishing or click-jacking attacks [8, 25]. Other prior work suggested content-based and heuristic approaches for protecting mobile users from phishing websites [68, 57, 30]. To my knowledge, none of these solutions have been implemented and phishing remains a problem.

This work is relevant to my dissertation as it shows that phishing attacks continue to work on mobile devices. In particular, Tuncay et al.’s results showing that phishing attacks for permissions were successful on Android suggest that the credential phishing attack described in my dissertation is also likely to be successful.

Clipboard-Based Password Managers

In 2013, Fahl et al. [23] analyzed 21 password managers on Android and showed that they encouraged users to copy passwords to their clipboard, which can be accessed by any app on the device. Zhang et al. [73] showed that the clipboard could be used for both code injection and phishing attacks. While most password managers no longer use the clipboard as their primary means for autofill, several popular password managers are still clipboard-only and even those that use other autofill approaches allow users to copy and paste their credentials as a backup method if the primary autofill method fails. Additionally, while background access to the clipboard is prohibited in the latest versions of both Android and iOS, home screen widgets are still able to access this content when in the foreground, meaning that this is still a valid attack vector [6].

WebView

Yang et al. [70] showed that untrusted iframes inside a WebView can be used to break web messaging integrity, stealthily access sensitive mobile functionalities, and perform phishing attacks.

And Tuncay et al. [63] designed and tested Draco, a uniform access control framework to protect apps from web code running in Android WebView. While embedded browsers in Android apps (WebViews) can execute web code and provide JavaScript bridges that give web code access to internal app code, they have no means of providing origin-based access control to these JavaScript bridges. Draco solves this problem by providing developers with the ability to implement strict policies about which app functionality can be accessed from web code. Draco would not protect against either of the WebView based attacks I describe in this dissertation because the attack that leverages communication between the app and the WebView assumes a malicious app and the other attack does not require such communication. Like Luo et al. [40], Yang et al. [70] did not leverage autofill in their attacks.

Attacks on the Accessibility Service

Fratantonio et al. [28] showed how accessibility permissions could be used to gain complete control over the UI feedback loop, allowing the attacker to steal user credentials. They suggest that this vulnerability is made worse by the fact that apps do not make users aware of the permissions requested by an application on installation. Jang et al. [36] identified twelve attacks against accessibility support services in both the iOS and Android sandboxes. Other work has noted how accessibility services expose private information [38] and allow for keylogging attacks [46]. Because most password managers on Android still request accessibility service permissions for backwards compatibility and autofill in the browser, vulnerabilities related to permissions in this service are relevant to my work.

2.2.3 Usability and User Studies

Prior password manager user studies focused largely on adoption [1, 5] or password strength and reuse [41], and targeted either desktop [14, 72] or mobile devices [56, 1]. In addition,

none of these studies involved observing users as they operated their password manager on their own devices, instead relying solely on interview, survey, or post-study questionnaire data. My work is the first based on a semi-structured observational interview of password manager use across devices, allowing us to identify issues not found by prior work, such as the frequent use of multiple managers and problems syncing across devices. Observation also turned out to be important because several users initially misreported which password manager they used and struggled to convey in words issues they experienced during use.

User Studies of Password Managers on the Desktop

Prior user studies of desktop based password managers focus on the impact of managers on password strength and adoption. Lyastani et al. [41] found that password managers do lead to the creation of stronger passwords overall, but that users still rarely maintain strong, unique passwords for all of their accounts. And Pearman et al. [72] conducted a semi-structured interview of 30 individuals, including users and non-users of managers, to better understand adoption and usage, concluding that while users of browser-based managers are more driven by convenience, users of separately installed managers are more driven by security. This dissertation helps explain the findings of both Lyastani et al. [41] because I found that generated passwords present usability hurdles to users, such as cross-device entry. My work also helps explain the key finding of Pearman et al. [72] because I found that users of browser-based managers often start using them without any forethought because they persistently pop up, whereas users of third-party managers adopt them at work or based on recommendations.

Pearman et al.'s findings nuanced those of Fagan et al. [22], who found that non-users of managers avoid them due to security concerns, while users adopt managers for reasons of convenience. More recently, Ray et al. [52] replicated Pearman et al.'s protocol with 26 adults over the age of 60, finding that older adults are more likely to adopt a manager if the recommendation comes from a family member and have more concerns with storing passwords in the cloud and syncing than younger adults. And in 2006, Chiasson et al. [14] found that users' incomplete mental models led to vulnerabilities in their studies of two desktop-based managers.

User Studies of Password Managers on Mobile Devices

Compared to password managers used on desktop/laptops, there is little work in the usability space focused on mobile managers. Akaldi et al. [1] examined manager reviews in the app store for both Google and Android devices, supplemented by a user survey, and found that “no perceived usefulness” and lack of enjoyment, possibly due to finding them difficult to use, deterred users from adopting them. These findings comport with those of Arias et al. [4].

More recently, Seiler-Hwang et al. [56] analyzed the usability of four smartphone password managers, with 20 users evaluating each by first completing a series of 7 tasks meant to represent the main functionality of a PM and then filling out post-task questionnaires. The 7 tasks were based on the work of Chiason et al. [14], and included initialization, migration, login, app interaction, updating a password, and modifying security settings. As noted by Seiler-Hwang et al. [56] when they discussed limitations, I believe that their task-based approach may have altered user behavior, specifically as it relates to updating security setting. They found that only 7.5% of users left default settings unaltered, whereas when I actually observed users I found that an overwhelming majority of users do not modify default settings at all, making it even more critical for settings to be secure by default. However, I recognize that there may be other explanations for this difference other than methodology, such as the fact that they focused exclusively on mobile managers.

2.2.4 Relation to This Work

To my knowledge, my work is the first to study the strength of password generators in password managers and the first to simultaneously consider the full password manager lifecycle [15] (i.e., generation, storage, and autofill). Much of the work examining the security of password manager autofill and storage on the desktop is now over five or more years old [58, 60, 39, 29]. As there have been significant updates to password managers in that time, I have replicated this early work to determine whether the password managers I studied have addressed the core issues revealed by this prior work, or whether they remain vulnerable.

My research also makes several unique contributions regarding mobile password managers. I am the first to analyze the security of password managers on iOS. Second, I leverage the

same tests I ran on the desktop and apply them to mobile password managers, the first time these types attacks have been tested on mobile platforms. Third, I present and evaluate novel phishing attacks against WebView on both iOS and Android. Fourth, I replicate and expand the work of Aonzo et al. [3] on Android, demonstrating that the lack of a good method for mapping which credentials should be used with which applications remains a significant problem. Compared to Aonzo et al.’s work, I study a wider range of password managers (thirteen in my study versus five in theirs).

Finally, based on the results of my mobile and desktop analysis, I conduct the first observational interview investigating third party password manager usage. In contrast to my work, the interviews conducted by Pearman [50] only included seven users of third party password managers and had no observational component. And the work of Lyastani [41] focused specifically on the effect of password managers on password strength, rather than usage in general. My findings help explain the problems with reuse identified by Lyastani [41] and Pearman’s [50] finding that users of browser-based manager are more convenience oriented.

Chapter 3

Browser-Based Password Managers on the Desktop

In this section¹, we provide a comprehensive evaluation of browser-based password managers, including five browser extensions and six password managers integrated directly into the browser. We also include two desktop clients for comparison. Five or more years have passed since the previous studies that examined password managers in the browser [39, 58, 60], leaving it unclear whether password managers remain vulnerable or whether they are now ready for broad adoption. To answer this question, we update and expand on these previous results and present a thorough, up-to-date security evaluation of thirteen popular password managers.

In our evaluation, we consider the full password manager lifecycle [15]—password generation (Section 4.6), storage (Section 4.7), and autofill (Section 3.4). For password generation, we evaluate a corpus of 147 million passwords generated by the studied password managers to determine whether they exhibit any non-randomness that an attacker could leverage. Our results find several issues with the generated passwords, the most severe being that a small percentage of shorter generated passwords are weak against online and offline attacks (shorter than 10 characters and 18 characters, respectively). We also replicate earlier work examining the security of password storage [29] and autofill [39, 58, 60].

¹This chapter is adopted from my publication: Oesch, Sean, and Scott Ruoti. "That Was Then, This Is Now: A Security Evaluation of Password Generation, Storage, and Autofill in Browser-Based Password Managers." USENIX Security Symposium. 2020.

Our results find that while password managers have improved in the past five years, there are still significant security concerns. We conclude the section with several recommendations on how to improve existing password managers as well as identifying future work that could significantly increase the security and usability of password managers generally (Section 5.4).

Our **contributions** include:

1. Our research finds that app-based and extension-based password managers have improved security compared to five years ago. However, there are still residual vulnerabilities that need to be addressed—for example, several tools will automatically fill passwords into compromised domains without user interaction and others that do require user interaction allow users to disable it. As such, it is important to both carefully select a password manager and to configure it properly, something that may be difficult for many users.
2. To our knowledge, this work is the first evaluation of password generation in password managers. As part of this evaluation, we generated 147 million passwords representing a range of different password managers, character composition policies, and length. We evaluated this corpus using various methods (Shannon entropy, χ^2 test, zxcvbn, and a recurrent neural net) to find abnormalities and patterns in the generated passwords. We found several minor issues with generated passwords, as well as a more serious problem where some generated passwords are vulnerable to online and offline attacks.
3. Our work is the most comprehensive evaluation of password manager security to date. It studies the largest number of password managers (tied with Gasti and Rasmussen[29]) and is the only study that simultaneously considers all three stages of the password manager lifecycle [15]—password generation, storage, and autofill (prior studies considered either storage or autofill, but not both simultaneously).
4. Prior security evaluations of password managers in the literature are now five or more years old. In this time, there have been significant improvements to password managers. In our work, we partially or fully replicate these past studies [29, 39, 58, 60] and demonstrate that while many of the issues identified in these studies have been

addressed, there are still problems such as unencrypted metadata, unsafe defaults, and vulnerabilities to clickjacking attacks.

3.1 Analyzed Password Managers

In this work, we analyzed 13 different password managers. These password managers can be categorized based on their level of integration with the browser: app, extension, and browser. We focused on password managers in the browser but included two desktop clients for comparison. Apps are desktop clients that are not integrated with the browser. Extension-based password managers are deployed as a browser extension and do not rely on a desktop application. Browser-based password managers are native components implemented as part of the browser. We chose from among the most popular systems within each of these categories.

The breakdown of analyzed password managers into these categories is given in Table 3.1. This table also reports on features related to utility and usability—support for password generation and autofill, support for synchronizing extension settings and password vaults using the cloud, ability to use the password manager from a command line interface—as well as security—whether the tool supports multi-factor authentication (MFA), whether the password vault can be locked, whether the master password for the vault must be entered on its own tab or application (to prevent spoofing of this dialog [12]), whether the password manager provides a tool to assess the security of stored accounts and passwords, whether the manager clears passwords from the clipboard after they are copied, and whether the tool is open source.

In the remainder of this section, we discuss each password manager analyzed and indicate which version of the password manager we evaluated. In-depth details regarding password generation, autofill, and storage are found in their respective sections.

3.1.1 App

The app-based password managers we analyzed eschew cloud syncing of vaults and settings in favor of manual synchronization to increase security.

Table 3.1: Analyzed Password Managers

System		Supports generation Supports autofill Cloud sync for extension settings Cloud sync for vault CLI support Supports MFA Lockable Vault Login on separate tab or app Has assessment tool Clears clipboard Open Source											
App	KeePassX	●	○	○	○	○	○	○	●	●	○	●	●
	KeePassXC	●	●	○	○	●	○	○	○	○	○	●	●
Extension	1Password X	●	●	○	●	●	●	●	○	●	○	○	○
	Bitwarden	●	●	○	●	●	●	○	○	○	○	○	●
	Dashlane	●	●	◐	●	○	○	○	○	●	○	○	○
	LastPass	●	●	○	●	●	●	○	○	●	○	○	○
	RoboForm	●	●	○	●	○	○	○	○	○	○	○	○
Browser	Chrome	◐	●		●	○	○	○	○	●	○		◐
	Edge	○	●		●	○	○	○	○	○	○		○
	Firefox	○	●		●	○	○	○	○	○	○		●
	IE	○	●		●	○	○	○	○	○	○		○
	Opera	○	●		●	○	○	○	○	○	○		◐
	Safari	◐	●		●	○	○	○	○	○	○		◐

● Secure behavior ◐ Partially secure behavior ○ Insecure behavior

KeePassX (v2.0.3). KeePass is an app-based password manager originally built using the .NET platform and intended for use on Windows. KeePassX is a cross-platform port of KeePass, replacing the .NET platform with the QT framework.

KeePassXC (v2.3.4). KeePassXC is a fork of KeePassX intended to provide more frequent updates and additional features not found in KeePass or KeePassX (e.g., more options for password generation, a command line interface). KeePassXC also provides a browser extension that interfaces with the app to autofill passwords in the browser. In total, the KeePass family of applications is estimated to have 20 million users [21].

3.1.2 Extension

Extensions lack permissions to clear the clipboard and so none of the extension-based password managers support this feature, leaving user passwords vulnerable to any application with clipboard access indefinitely. None of the extensions we analyzed supported synchronizing settings for the extension itself, requiring that users remember to correctly update these settings to match their security preferences for each new device they set up. These extension settings include security critical options, such as whether to log out when the browser is closed, whether to use autofill, and whether to warn before filling insecure forms. The user experience for each of the extension-based password managers is mostly similar.

1Password X (v1.14.1). 1Password is estimated to have 15 million users [21]. 1Password provides both an app-based client (1Password) and an extension-based client (1Password X); in this work, we evaluated the extension-based client because it is the recommended tool if integration with the browser is desired (something we assume most users would want).² While the security of both systems is similar, there are a few small differences—e.g., the password is cleared from the clipboard in the app, but not the extension. Unique to 1Password, to initially download the password vault from the cloud it is necessary to enter a 128-bit secret key that was presented to the user when they generated their account, providing an extra layer of security to the cloud-based password vault.

²<https://support.1password.com/getting-started-1password-x/>

Bitwarden (v1.38.0). Bitwarden is unique within the extension-based password managers that we analyzed in that all of its functionality is available to non-paid accounts, whereas other password managers required a subscription to gain access to some features.

Dashlane (v6.1908.3). Dashlane is estimated to have 10 million users [21]. In addition to storing the username and password for each website, Dashlane also tracks and synchronizes the following three settings on a per-site basis: “always log me in”, “always require [the master password]”, and “Use [password] for this subdomain only.” This feature provides a slight advantage when compared to other extension-based password managers that do not synchronize any extension settings.

LastPass (v4.24.0). LastPass is estimated to have 16.5 million users [21], the most of any commercial password manager.

RoboForm (v8.5.6.6). RoboForm is estimated to have 6 million users.³ Like 1Password, RoboForm offers both an app-based client and an extension-based client; in this work, we evaluated the extension-based client for the same reason we took this approach with 1Password X.

3.1.3 Browser

Compared to both app-based and extension-based password managers, browser-based password managers lack many features. While all browser-based password managers allow the cloud account storing the password vault to be protected using multi-factor authentication, none except Firefox enable this vault to be locked short of removing the account from the browser. Firefox provides the option to use a master password to restrict access to the password vault. As these password managers do not have settings to sync and never copy a password to the clipboard, those features are not applicable.

Chrome (v71.0). Chrome has some support for generating passwords. It detects when a user might need a password and offers to generate a password for the user. Unlike any other password manager, Chrome has basic functionality to try to detect the password policy.

Edge (v42.17134). Firefox (v64.0). Internet Explorer (v11.523). Opera (v58.0.3135). These password managers are all similar in high-level functionality.

³<https://www.roboform.com/business/features>

Safari (v12.0). Safari can generate passwords when integrated with iCloud Keychain, though these passwords are always of the form “xxx-xxx-xxx-xxx”.

3.1.4 Updates for Password Managers

Since we conducted our research, there have been some minor changes in several of the password managers: (1) KeePassXC has transitioned to using Argon2D as their default key derivation function, (2) LastPass has updated their password generation interface, removing the option to select the number of digits, and (3) RoboForm has updated their password generation interface, removing the option to select the number of digits and increasing the default password length to 16. We are also aware of a couple more significant changes on the horizon: Firefox will transition to using Firefox Lockbox as its default password manager, and Edge will transition to being built on top of the Chromium project.

3.2 Password Generation

Password generation is the first step in the password manager lifecycle. Of the 13 password managers in our evaluation, seven have full support for password generation—KeePassX, KeePassXC, 1Password X, Bitwarden, Dashlane, LastPass, and Roboform—and two have partial support—Chrome and Safari. To provide a baseline by which to compare the password managers, we wrote a python script that generates passwords using `/dev/random` and the online Secure Password Generator⁴ (SPG), the first search result when searching for “password generator” on Google.

3.2.1 Settings and Features

Table 3.2 provides a summary of configuration options, default settings, and features for each of the tools tested. All password managers support ensuring that at least one character from each selected character set is included in the generated password, though this can be turned off in KeePassX, KeePassXC, and LastPass. All password managers other than the

⁴<https://passwordsgenerator.net>

Table 3.2: Overview of Password Generation Features

System	Abbreviation	Supported lengths	Require diverse characters	Avoid difficult characters	Default length	Default composition	Preserve safe settings	Symbol set
KeePassX	kpx	3–64	●	●	16	ld	●	!"#\$%&'()*+,-./:;<=>?@[\]^_`{ }~
KeePassXC	kpxc	1–128	●	●	16	ld	○	!"#\$%&'()*+,-./:;<=>?@[\]^_`{ }~
1Password X	oneps	8–50	●	●	20	all	○	!#%)*+,-.:=>?@]^_}~
Bitwarden	bw	5–128	●	●	14	ld	○	!#\$%&*@~
Dashlane	dlan	4–28	●	●	12	all	●	!"#\$%&'()*+,-./:;<=>?@[\]^_`{ }~
LastPass	lpass	4–100	●	●	12	ld	○	!#\$%&*@~
RoboForm	robo	1–99	●	●	14	all	○	!#\$%&*@~
Chrome	chrom	> 1	●	○	15	all		!-._:_
Safari	sfri	15	●	○	15	all		-
SPG	psgn	6–2048	●	●	16	all		!"#\$%&'()*+,-./:;<=>?@[\]^_`{ }~
/dev/random	dvrn	> 1	○	○				!"#\$%&'()*+,-./:;<=>?@[\]^_`{ }~ _

● Secure behavior ● Partially secure behavior ○ Insecure behavior

browser-based password managers also have an option to avoid generating passwords that contain characters that may be difficult for users to read and/or memorize (e.g., hard to pronounce, looks similar to another character), though the exact characters removed are not consistent between password managers.

While all password managers support the same set of letters and digits ([A-Za-z0-9]), they each had different symbol sets. KeePassXC had the largest symbol set, supporting all standard ASCII symbols (other than space) as well as supporting the extended ASCII symbol set. KeePassX and Dashlane also support the standard ASCII symbols (other than space), but not the extended ASCII symbol set. 1Password supports just over half of the ASCII symbols (19 symbols), with the other systems supporting 8 or fewer symbols. As expected, limiting the symbol set has a significant impact on the strength of generated passwords, the implications of which are discussed later in this section.

One issue common in most password managers is that they save the last used settings as the new default settings. While this might seem like a feature targeted at usability, it has the potential to cause users to use less than optimal settings when generating passwords. In general, there are two reasons for users to change their password generation settings: (1) establishing safe default settings, (2) generating a password that conforms with a policy that is weaker than the default settings. In the latter case, the newer, weaker settings will replace the older, stronger settings as the new defaults. While users can manually restore their safer settings, there is no guarantee that they will do so. Dashlane takes the optimal approach by not automatically saving the latest settings but giving the user the option to override the current defaults. KeePassX takes a middle-of-the-road approach, saving the new settings for future passwords generated until the application is closed and opened again.

3.2.2 Password Collection and Analysis

To evaluate the quality of passwords generated by the password managers, we first collected a large corpus of generated passwords from each password manager. We use a variety of methods to generate passwords: existing command line interfaces (Bitwarden, our python tool), modifying the source code to add a command line interface (Chrome, KeePassX, KeePassXC), or using Selenium (1Password X, Dashlane, LastPass, RoboForm). We were

unable to analyze passwords for Safari as it does not have any mechanism for scripting password generation, though we did manually generate and analyze 100 passwords to check for any obvious problems and did not detect any.

Generation was parameterized by character classes—letters (l), letters and digits (ld), letters and symbols (ls), symbols and digits (sd), and all four classes together (all)—and password length—8, 12, and 20 characters long—in order to determine if these options had any effect on the randomness of generated passwords. Most tools defaulted to requiring that generated passwords contain one character from each character set, with only Chrome, KeePassX, KeePassXC, and our python tool not having this option enabled. For each password manager, character class, and password length we generated 1 million passwords, except 1Password X which does not allow passwords to be generated that only have symbols and digits. This resulted in a corpus of 147 million passwords ($10 \times 5 \times 3 - 3$).

After collecting this data set, we analyzed its quality in terms of randomness and guessability. There is no known way to prove that a pseudorandom generator is indistinguishable from random, so instead we leveraged a variety of analysis techniques, each attempting to find evidence of non-random behavior: Shannon entropy, χ^2 test for randomness, the zxcvbn password analysis tool [67], and a recurrent neural net-based password guesser [44].

Shannon entropy is used to check for abnormalities in the frequency of characters (not passwords) produced by each generator. The Shannon entropy of a set is a measure of the average minimum number of bits needed to encode a string of symbols based on the frequency of their occurrence. It is calculated as $-\sum_i p_i \log_b(p_i)$. While Shannon entropy is a bad measure for user-chosen passwords [10], it is useful in evaluating the relative strength of random passwords. Shannon entropy is not affected by the length of passwords, only by the number of distinct characters that can be present in a string and their relative frequency within the corpus.

The χ^2 test for randomness is a simple statistical test for determining whether the difference between two distributions can be explained by random chance. We used the χ^2

test to evaluate each of our passwords sets independently and corrected our p-values using a Bonferonni correction⁵ to account for the multiple statistical tests from the same family.

The `zxcvbn` tool created by Daniel Wheeler [67] is used to detect dictionary words and simple patterns that might be present in passwords, both potential examples of non-randomness. `zxcvbn` also estimates the number of guesses a password cracker would take to break a password, which we use to understand if passwords are resilient to online and offline guessing.

In order to detect whether generated passwords had more subtle patterns than what `zxcvbn` could detect, we used the neural network password analyzer built by Melicher et al. [44]. This analyzer uses a Long Short-Term Memory (LSTM) recurrent neural network (RNN) architecture to build a password guesser based on a training set. As output, it produces a Monte Carlo estimation of how long it would take the trained password guesser to guess passwords in a test set. The configuration files we used for training and testing are provided in Listing 3.1 below. For each password corpus, we used 80% of the passwords to train the neural network and tested against 20% of the passwords. Due to problems with the analyzer, we were only able to test passwords of length 8 and 12, as length 20 passwords would crash with an out of memory exception regardless of what settings were used.

While `zxcvbn` and the recurrent neural net are both used to evaluate the quality of randomness in the generated passwords, they also served to give approximations for how many guesses it would take for an online or offline guessing attack to try that password. Passwords that require more than 10^6 guesses are considered to be resilient against online attacks and passwords that require more than 10^{14} guesses are considered to be resilient against offline guessing [27]. Using this guess count, we were able to analyze whether the password managers were generating passwords that were vulnerable to these attacks.

⁵To represent this correction, all p values are multiplied by 147, with a maximum value of 1.00. For this reason, most p values reported are 1.00, as only clearly significant results stay significant with such a large correction.

```

1 {
2   "args": {
3     "pwd_file": [
4       "$TRAINING_FILE"],
5     "pwd_format": ["list"],
6     "log_file": "$LOG_FILE",
7     "arch_file": "$ARCH_FILE",
8     "weight_file":
9       "$WEIGHT_FILE"
10  },
11  "config": {
12    "intermediate_fname":
13      "$INTERMEDIATE_FILE",
14    "min_len":
15      $PASSWORD_LENGTH,
16    "max_len":
17      $PASSWORD_LENGTH,
18    "training_chunk": 1024,
19    "layers": 2,
20    "hidden_size": 1000,
21    "dense_layers": 1,
22    "dense_hidden_size": 512,
23    "generations": 5
24  }
25 }

```

```

1 {
2   "args": {
3     "enumerate_ofile":
4       "$GUESSES_FILE",
5     "log_file": "$LOG_FILE",
6     "arch_file": "$ARCH_FILE",
7     "weight_file":
8       "$WEIGHT_FILE"
9   },
10  "config": {
11    "guess_serialization_method":
12      "delamico_random_walk",
13    "password_test_fname":
14      "$TESTING_FILE",
15    "parallel_guessing": true,
16    "intermediate_fname":
17      "$INTERMEDIATE_FILE",
18    "min_len": $PASSWORD_LENGTH,
19    "max_len": $PASSWORD_LENGTH,
20    "training_chunk": 1024,
21    "layers": 2,
22    "hidden_size": 1000,
23    "dense_layers": 1,
24    "dense_hidden_size": 512,
25    "generations": 5
26  }
27 }

```

Listing 3.1: Neural Network Configuration—Training (Left) and Testing (Right)

3.2.3 Results

Password Strength: Our analysis of the generated passwords found that nearly all passwords of length 12 and longer were sufficiently strong to withstand both online and offline guessing attacks (see Figures 3.1c and 3.1d). Still, not all password managers created passwords of equal strength, with these small perturbations having a significant effect on the percentage of length 8 passwords that were secure against offline guessing attacks (nearly all were secure against online guessing attacks) (see Figures 3.1a and 3.1b). These differences in strength can largely be explained by the different composition of character set classes used by each of the password managers. While the difference is most pronounced when considering symbols (see Table 3.2), several password managers also limit the available letters and digits (e.g., removing ‘0’ and ‘O’ due to similarity). Looking at character frequencies (see Table 3.3), we also found that Dashlane uses a different set of letters depending on the length of the passwords; it is unclear why Dashlane exhibits this behavior.

Randomness: Our χ^2 testing found several instances of non-random behavior in the generated passwords (see Table 3.4, detailed χ^2 and p values are in Tables 3.5–3.7 below). All but one of the non-random character frequency distributions can be explained by a single feature—requiring that passwords have at least one character from each character set. When this feature is not enabled, the probability that any given character will appear in a password is proportional to the length of the password, and the number of characters from all the enabled character sets (see Equation 3.1). When this feature is enabled, the probability is also proportional to the number of characters in that character set (see Equation 3.2), causing character frequencies to be higher for characters that come from smaller character sets (e.g., digits, symbols), explaining the non-uniformity detected by the χ^2 test.

$$length * \frac{1}{|characters_{all}|} \tag{3.1}$$

$$((length - |sets|) * \frac{1}{|characters_{all}|}) + \frac{1}{|characters_{set}|} \tag{3.2}$$

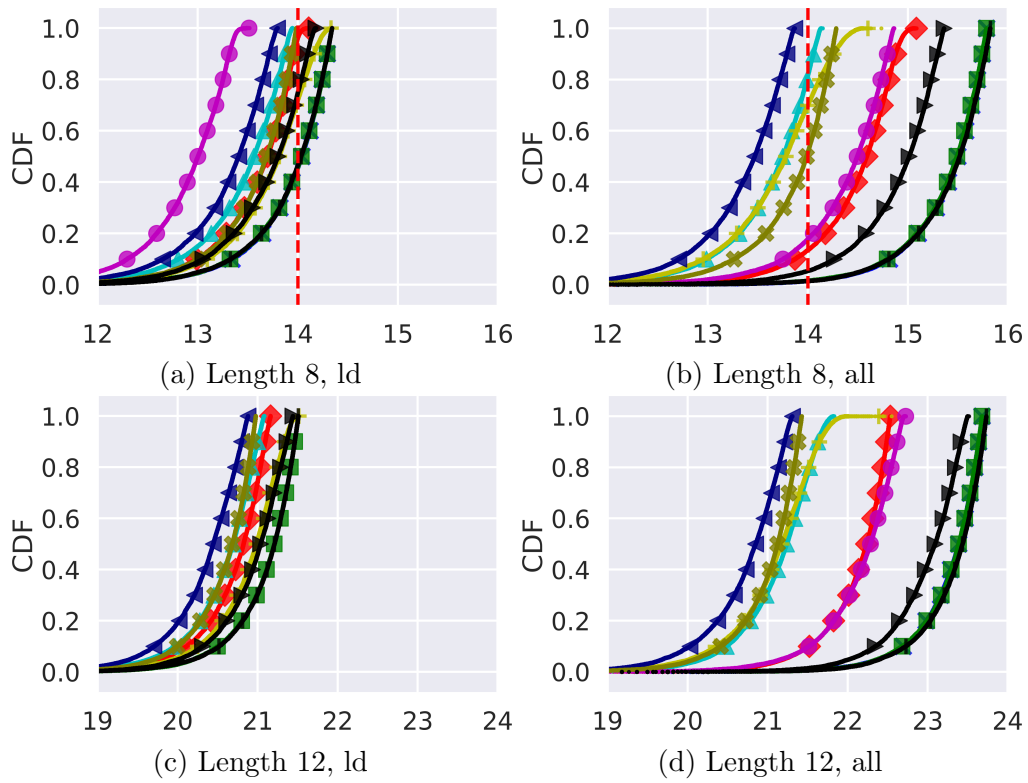
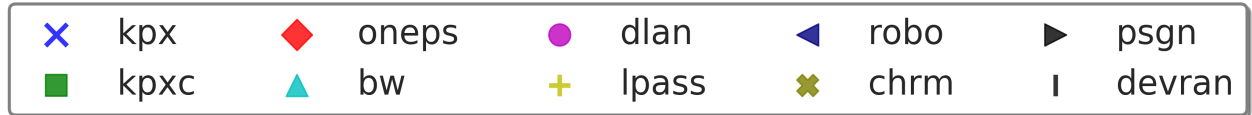


Figure 3.1: Neural Network Guess Estimates ($\log_{10}$). Differences are primarily attributed to character set size.

Table 3.3: Character Frequencies for length 20 passwords using all characters. Groups of similar characters represent a requirement to include at least one character from that set, causing characters from smaller sets to be selected with greater frequency.

System	Characters Sorted by Frequency
kpx	'+,7lFr[AE/8"\$0dNzGMn`_*3;D:i Z@s=#]whRb6~&Wm(2ck)\g^oy<aL}JCTq4e !->VI1BPvY9HSUjp{?5%xt0fX.uQK
kpxc	NtpgT@v0<Be1hiY)H-`Kk;IXu^c4z\$yqo6F/r>S_%Z3+U[=DL\as"0(2'VA?PdRm .:*jb]W~}Exn{f!Q 7#CJw8G,&9lM5
oneps	0314569782>^:.*@.-~+%?,_V=a}N)!d)YjZK#ubeCATJUGBEDyozrgkMRtHwLvXWmqxf QhsniPFpc
bw	%7#9532~@46!\$&8*IYBomtbJFLUPVnXdzSexagHZrwusiMkpqcWNRvQKhfDCGAjyTE
dlan	5473698QRHDNPAFMBKSCLYTXEGJijepnfgtryhbdkmqsxacz_*@~={ '[;&, #.: "/\$ (^ +}-%] ?o`><\wuU2WvVZ
dlan*	3498576QNBHPAFMXJCYKTGSDRLEdqzmnpsfjghbtckaioyre/\$# {!<- ,?"(\=] .~*^'+` :;}>_) [%@&
lpass	%!\$#&*@~jAGfRMOYPobszleTUilwVhtDKNQqJgBSaWmpudcnLkEyHrZFxCXv398754 2160
robo	%#@!^\$8624375HLYJXPDFCWAUENKVSTiRQGBMydgstkvqpfnjbwaemhrucx9Zz
chrm	umSHDMeYNbnEGzCwaspZg6f:!XqLTBWrR9t5h3JP8Q7jc_iAFVK-kdxv2Uy.4
psgn	4239750618LoQPYliRHpJkqIUZOnWBxmNhvdDbgAXtuVcwzysSCarMj EGKTfeF\!/(. +%}@ '=[\${?:*>&)^~ -; " ^] ,<#_
dvrn	. \zdAP4L0^W,6@&+3w%?ebSqc-"Y\$8EM'~QVu}iGojv(tK:y;I>#<TD_aU9C[1 rH)/h5Z1_ sR`=m0]{*xXgnBNpfFJk2!7

*Length 12 passwords. Dashlane uses different characters sets for long and short passwords.

Table 3.5: Length 8 χ^2 Scores for Character Frequency

System	all		l		ld		ls		sd	
	p	χ^2	p	χ^2	p	χ^2	p	χ^2	p	χ^2
KeePassX	1.00	84.62	1.00	42.15	1.00	65.49	1.00	77.38	1.00	38.81
KeePassXC	1.00	85.16	1.00	67.35	1.00	61.41	1.00	76.88	1.00	35.27
1Password X	0.00	294756	1.00	41.80	0.00	132469	0.00	17747		
Bitwarden	0.00	724697	1.00	53.40	0.00	361209	0.00	362807	1.00	12.54
Dashlane	0.00	729301	0.00	1203	0.00	334844	0.00	47489	0.00	348990
LastPass	0.00	640316	1.00	72.20	0.00	96928	0.00	390413	0.00	156327
RoboForm	0.00	1108211	0.00	10792	0.00	470973	0.00	605343	0.00	41584
Chrome	1.00	54.95	1.00	38.50	1.00	47.51	1.00	40.28	1.00	16.16
SPG	0.00	445079	1.00	45.67	0.00	245539	0.0	10804	0.0	190506
/dev/rand	1.00	77.65	1.00	59.37	1.00	62.17	1.00	89.01	1.00	37.73

Table 3.6: Length 12 χ^2 Scores for Character Frequency

System	all		l		ld		ls		sd	
	p	χ^2	p	χ^2	p	χ^2	p	χ^2	p	χ^2
KeePassX	.65	87.09	.74	44.12	.03	84.43	.45	83.96	.11	52.57
KeePassXC	.052	116.17	.44	51.78	.56	58.64	.65	77.46	.54	39.42
1Password X	0.00	95480	.54	45.44	0.00	33175	0	1600		
Bitwarden	0.00	481688	.49	48.60	0.00	239474	0.00	241181	.21	19.20
Dashlane	0.00	487295	0.00	765	0.00	224131	0.00	32113	0.00	233758
LastPass	0.00	428916	.73	44.30	0.00	64703	0.00	258080	0.00	104851
RoboForm	0.00	738458	0.00	7277	0.00	312865	0.00	403972	0.00	27661
Chrome	.70	53.71	.51	46.11	.15	65.53	.99	31.27	0.00	34.3
Web generator	0.00	297694	.047	69.07	0.00	163675	0.00	7289	0.00	125531
/dev/rand	.33	99.23	.27	56.73	.75	53.10	.31	89.93	.55	40.11

Table 3.7: Length 20 χ^2 Scores for Character Frequency

System	all		l		ld		ls		sd	
	p	χ^2	p	χ^2	p	χ^2	p	χ^2	p	χ^2
KeePassX	.62	88.10	.91	38.06	.14	73.07	.11	98.34	.30	45.11
KeePassXC	.49	92.57	.79	42.76	.97	41.66	.71	75.38	.92	28.97
1Password X	0.00	12789	.82	38.21	0.00	2367	.03	90.32		
Bitwarden	0.00	289893	.72	42.84	0.00	143389	0.00	143720	.21	19.10
Dashlane	0.00	956201	0.00	443060	0.00	401737	0.00	822537	.17	42.48
LastPass	0.00	256787	.50	50.32	0.00	38336	0.00	156177	0.00	63559
RoboForm	0.00	442762	0.00	4524	0.00	188292	0.00	241760	0.00	16928
Chrome	.91	46.01	.36	49.88	.25	61.8	.50	51.2	.056	20.60
Web generator	0.00	178091	.69	45.53	0.00	98651	0.00	4617	0.00	75043
/dev/rand	.63	88.73	.22	58.42	.29	66.77	.24	92.88	.49	41.62

While the results for Bitwarden (sd) and Dashlane (l) may at first not appear to follow this pattern, they in fact do. Bitwarden (sd) has equal numbers of symbols and digits (see Table 3.3, causing them to be selected with equal frequency. In contrast, Dashlane (l) has a non-random distribution because it uses a different number of upper and lowercase letters.

The only non-random result that cannot be explained at least partially by this feature is RoboForm (l), which has an equal number of upper and lowercase characters. Looking at all the character frequencies for RoboForm, we find that uppercase letters, other than ‘Z’, are selected more frequently than the lowercase letters. Additionally, the characters ‘Z’, ‘z’, ‘9’ are consistently the least frequently selected characters. While it is not entirely clear what causes this issue, we hypothesize that it might be related to selecting characters using modular arithmetic (e.g., $rand() \% (max - min) + min$), which can have a slight bias to lower valued results.

Random but Weak Passwords: In our analysis of the zxcvbn results, we found that occasionally all password managers would generate exceptionally weak passwords, examples of which are shown in Table 3.8. While this is expected behavior for a truly random generator, it still results in suboptimal passwords.

Even though randomly generated length 8-character passwords have the potential to be resilient to offline attack (e.g., $\log_{10}(96^8/2) = 15.56$), password managers will present users with passwords of this length that are vulnerable to both online and offline attacks. At length 12, the weakest passwords are no longer vulnerable to online attacks but are still vulnerable to offline attacks. Finally, at length 20 the weakest passwords were able to withstand an offline attack. While the occurrence of these weak passwords is relatively rare (less than 1 in 200), it is still preferable to choose passwords of sufficient length such that even randomly weak passwords are likely to be resilient to online and offline attacks. Based on our analysis of these results, that is length 10 for resilience to online attacks and length 18 for resilience to offline attacks.

Table 3.4: χ^2 test for random character distribution

System	l	ld	ls	sd	all
KeePassX	✓	✓	✓	✓	✓
KeePassXC	✓	✓	✓	✓	✓
1Password X	✗	✓	✗	✗	
Bitwarden	✗	✓	✗	✗	✓
Dashlane	✗	✗	✗	✗	✗
LastPass	✗	✓	✗	✗	✗
RoboForm	✗	✗	✗	✗	✗
Chrome	✓	✓	✓	✓	✓
SPG	✗	✓	✗	✗	✗
/dev/rand	✓	✓	✓	✓	✓

✓ No statistically significant results (random)

✗ Statistically significant result (non-random)

Table 3.8: Randomly Generated Weak Passwords

System	Length Composition Guesses ($\log_{10}$)			Password
KeePassX	8	l	4.96	TaKEdeen
KeePassXC	8	sd	4.84	'+'+'+'+_+
1Password X	12	ls	8.76	oMMMMMT?m*m
Bitwarden	8	all	4.12	d@rKn3s5
Dashlane	8	sd	4.48	////\$8\$8
LastPass	12	all	8.92	B@KeRee22241
RoboForm	8	ls	5.02	SAWyE@rS
RoboForm	8	sd	4.06	2345678#
Chrome	8	all	4.85	Tz5a5a5a
SPG	8	ls	5.32	nW\$nW\$RR
/dev/rand	12	l	9.0	MrKNxQNDAViS

3.3 Password Storage

Password storage is the second stage of the password manager lifecycle. To evaluate the security of password storage, we manually examined the local password databases created by each password manager, looking to see what information was and was not encrypted, as well as examining how changes in the master password effected the encryption of data. We determined how encryption took place through a combination of claims from the password manager’s maintainer, options available in the client, and format of the ciphertext. We focus on the storage of the password vault on the local system as the cloud databases are not available to us for direct evaluation. An overview of this information is provided in Table 3.9.

3.3.1 Password Vault Encryption

The app-based and extension-based password managers all encrypt their databases using AES-256. These systems all use a key derivation function (KDF) to transform the master password (MP) into a cryptographic key that can be used for encryption. KeePassX and KeePassXC use AES-KDF with 100,000 rounds. All of the extension-based password managers, other than Dashlane, use PBKDF2, with only RoboForm using less than 100,000 rounds. Dashlane is the only password manager that uses a memory-hard KDF, Argon2D, with 3 rounds. While not used by default, KeePassXC does support the option of using Argon2D in place of PBKDF2.

Each of these password managers has different requirements for the composition of the master password. KeePass and KeePassX both allow any composition for the master password, including not using a master password at all. The extension-based password managers all require a master password but vary in composition requirements. LastPass, RoboForm, and Bitwarden require that the master password be at least eight characters but impose no other restrictions. 1Password X increases the minimum length to 10, but otherwise is the same as the other three. Only Dashlane has compositions requirements, requiring a minimum length of 8 characters and one character from each character class (lowercase, uppercase, digit, symbol).

Table 3.9: Overview of Password Vault Encryption

System	Storage	Encryption	KDF	KDF Rounds	Requires strong MP	Metadata Encrypted							
						URL	Icon	Username	Creation time	Modification time	Last use time	Fill count	Email Settings
KeePassX	File (.kbdx)	AES-256	AES-KDF	100,000	○	●	●	●	●	●	●	●	●
KeePassXC	File (.kbdx)	AES-256	AES-KDF	100,000	○	●	●	●	●	●	●	●	●
1Password X	File (.json)	AES-256	PBKDF2	100,000	●	●	●	●	●	●	●	○	○
Bitwarden	File (.json)	AES-256	PBKDF2	100,001	●	●	●	●	○	●	●	○	●
Dashlane	File (.aes)	AES-256	Argon2D	3	●	●	○	●	●	●	●	○	●
LastPass	File (.sqlite)	AES-256	PBKDF2	100,100	●	●	●	●	●	●	●	○	●
RoboForm	File (.rfo)	AES-256	PBKDF2	4,096	●	●	●	●	●	●	●	○	●
Chrome	File (.sqlite) ¹	OS				○	○	○	●	○	○	●	●
Edge	Windows Vault					●	●	●	●	●	●	●	●
Firefox	File (.json)	3DES	SHA-1	1	○	○	●	○	●	○	○	●	●
IE	Windows Vault					●	●	●	●	●	●	●	●
Opera	File (.sqlite) ¹	OS				○	○	○	●	○	○	●	●
Safari	OSX Keychain					●	●	●	●	●	●	●	●

¹On Linux, Chromium-based browser tries to store the password in the keyring or KWallet 4. If neither of these are available, it will store the passwords in plaintext [16].

● Secure behavior ● Partially secure behavior ○ Insecure behavior

Of the browser-based password managers, only Firefox handles the encryption of its password vault itself. It uses 3DES to encrypt the password data, using a single round of SHA-1 to derive the encryption key. It imposes no policy on the master password. Compared to other password managers that handle their own encryption, Firefox is by far the weakest.

The remaining browser-based systems rely on the operating system to help them encrypt the password vault. Edge, Internet Explorer, and Safari all rely on the operating systems keyring to store credentials. For Edge and Internet Explorer this is the Windows Vault; for Safari it uses the macOS keychain.

Chrome and Opera also rely on the operating system to encrypt the password, but how they do so varies by operating system. On Windows, the `CryptProtectData` function is used to have Windows encrypt the password with a key tied to the current user account. On Linux, these systems first try to write the password to the GNOME keyring or KWallet 4, falling back to storing the passwords in plaintext if neither of these keychains is available. On macOS, the passwords are encrypted with keys derived by the macOS keychain, though the website passwords themselves are stored locally rather than on the keychain.

Browser-based password managers, other than Firefox, rely on the operating system to encrypt passwords and therefore do not allow users to establish a master password. As such, there is no way to lock the password vault separately from locking the account. While outside the scope of this work, we also note that there is a need for more research examining the security of OS-provided encryption functions and keychains.

3.3.2 Metadata Privacy

Compared to earlier findings by Gasti and Rasmussen [29], we find that app-based and extension-based password managers are much improved in ensuring that metadata is properly protected. KeePassX and KeePassXC both encrypt all metadata. Extension-based password managers encrypt most metadata, but all have at least one item they do not.

1Password X stores extension settings in plaintext, allowing them to be read or modified by an attacker. These settings include security-related settings such as whether auto-lock is enabled, default password generation settings, and whether to show notifications. While Dashlane encrypts the website URLs, it does not encrypt the website icons it associates

with those URLs, allowing an attacker to infer websites for which a user has accounts. All extension-based password managers leak the email address used to log in to the password manager.

Browser-based managers that rely on an operating system provided keychain (Edge, Internet Explorer, Safari, as well as Chrome and Opera on Linux) use these tools to protect all relevant metadata. For the other browser-based password managers (Chrome and Opera on Windows and macOS, as well as Firefox on all operating systems), there is a significant amount of unencrypted metadata. All three of these password managers store the URL in cleartext, and only Firefox encrypts the username. They also reveal information about when the account was created, when it was last used, and how many times the password has been filled.

3.4 Password Autofill

Of the password managers we evaluated, only KeePassX did not support autofill in the browser⁶ and Bitwarden warns that its autofill functionality is experimental. To evaluate these tools, we developed websites that leveraged the attacks identified by Li et al. [39], Silver et al. [58], and Stock and Johns [60]. We also updated these attacks to address protections that have been added by browsers and password managers since the attacks were first described. Table 3.10 highlights several of our findings.

3.4.1 User Interaction Requirements

If an attacker can compromise a web page using either a network injection or XSS attack, they can insert malicious JavaScript that will steal the user’s password when it is entered. If a password manager autofills passwords without first prompting the user, then the user’s password will be surreptitiously stolen simply by visiting the compromised website. As such, user interaction should ideally be required before autofill occurs. Of the password managers we tested, only 1Password X and Safari always require user interaction before filling in

⁶There is a browser extension adding autofill for KeePassX, but it is a third-party tool not a part of the KeePassX project.

Table 3.10: Overview of Password Autofill Features

	Interaction Required for HTTPS Interaction Required for bad cert Interaction Required for HTTP	Won't fill same-origin iframe Won't fill cross-origin iframe	Won't fill different URL Won't fill HTTPS→bad cert Won't fill HTTPS→HTTP	Won't fill different action (static) Won't fill different action (dynamic) Won't fill different method Won't autofill different input fields	Won't fill type= 'text' field Won't fill non-login form fields	Fills password on transmission Obeys autocomplete= 'off'
System	Interaction	iframe	Difference in fill form		Fields	Misc
KeePassXC	● ● ●	● ○	○ ○ ●	● ● ○ ○	● ●	○ ○
1Password X	● ● ●	● ●	● ● ●	● ● ● ●	● ●	○ ○
Bitwarden	● ● ●	● ●	○ ○ ○	○ ○ ○ ○	● ○	○ ○
Dashlane	○ ○ ●	● ●	○ ○ ●	○ ○ ○ ○	○ ●	○ ○
LastPass	○ ○ ●	● ●	○ ○ ●	● ○ ○ ●	● ●	○ ○
RoboForm	● ● ●	● ●	○ ○ ○	○ ○ ○ ○	● ●	○ ○
Chrome	○ ● ○	○ ●	○ ● ●	○ ○ ○ ○	● ○	○ ○
Edge	○ ● ●	○ ●	○ ● ●	○ ○ ○ ○	● ○	○ ○
Firefox	○ ○ ○	○ ○	○ ○ ●	● ● ○ ○	● ○	○ ○
IE	○ ● ●	○ ●	○ ● ●	○ ○ ○ ○	● ○	○ ○
Opera	○ ● ○	○ ●	○ ● ●	○ ○ ○ ○	● ○	○ ○
Safari	● ● ●	● ●	● ● ●	● ● ● ●	● ●	○ ○

● Secure behavior ● Partially secure behavior ○ Insecure behavior

credentials. The remaining password managers exhibited different behavior depending on the protocol the website was served over (i.e., HTTPS or HTTP) as well as whether the HTTPS certificate was valid.

For websites served over HTTPS with a valid certificate, KeePassXC, Bitwarden, and RoboForm require user interaction by default, but also allow user interaction to be disabled. Dashlane, Lastpass, and Firefox default to autofilling passwords without user interaction, though there is an option to require user interaction. Chrome, Edge, Internet Explorer, and Opera always autofill user credentials. While having an option to require user interaction (Dashlane, LastPass, Firefox) is preferable to lacking that option (Chrome, Edge, Internet Explorer, Opera), in practice the results are likely the same for most users (who are unlikely to change their default options).

While network injection attacks are still possible on sites using HTTPS (i.e., TLS man-in-the-middle attacks [49]), they are much easier to accomplish and more likely if the HTTPS certificate is invalid. Reasons for a bad HTTPS certificate range from benign (e.g., expired by a day) to malicious (e.g., invalid signature, revoked). In both cases, password managers should altogether reject filling in the password or at the least require user interaction before autofilling the password. In the case of an invalid certificate, KeePassXC, Bitwarden, RoboForm, Dashlane, Lastpass, Firefox all function as they did with a valid certificate. Edge and Internet Explorer both change their behavior and always require user interaction for bad certificates. Chrome and Opera also change their behavior, entirely disabling the ability to autofill passwords.

Network injection attacks are also more likely and easier to accomplish when the website is served using an unsecured connection (i.e., HTTP). As with bad certificates, password managers should refuse to autofill the password or require user interaction before filling it in. KeePassXC, Bitwarden, and RoboForm continue to require user interaction by default, but do allow users to disable this requirement. Dashlane, LastPass, Edge, and Internet Explorer all change their behavior to always require user interaction before autofilling passwords on HTTP websites.

3.4.2 Autofill for iframes

Autofilling passwords within iframes is especially dangerous, regardless of whether user interaction is required or not [58, 60]. For example, clickjacking can be used to trick users into providing the necessary user interaction to autofill their passwords, allowing an attacker to steal passwords for vulnerable websites loaded in an iframe (same-origin or cross-origin). Even worse, if autofill is allowed for cross-domain iframes and user interaction is not required, then the attacker can programmatically harvest the user’s credentials for all websites where the attacker can perform a network injection or XSS attack (by loading compromised websites into iframes).

For both the clickjacking and harvesting attacks, the user must first visit a malicious website which will then launch the attacks, but this is often not a significant obstacle for an adversary. In the worst case, if a system is vulnerable to a harvesting attack and the attacker has access to the user’s WiFi access point (e.g., at hotel or airport)—allowing them to trivially conduct network injection attacks—then all of a user’s credentials can surreptitiously be stolen when the user views the network login page for the compromised access point [58, 60]

KeePassXC, 1Password X, Dashlane, and LastPass autofill within same-origin iframes, leaving them vulnerable to clickjacking attacks. Bitwarden and RoboForm also autofill within same-origin iframes, though if user interaction is required they are largely immune to clickjacking as this interaction happens outside of the website inside the extension drop-down. All of the browsers will autofill within a same-origin iframe.

KeePassXC does allow autofill for cross-domain iframes; while by default it does require user interaction before autofill in cross-domain iframes, this requirement can be disabled leaving KeePassXC vulnerable to the harvesting attack described above. Of the extension-based password managers, 1Password X, LastPass, and RoboForm will not fill autofill within a cross-origin iframe. Bitwarden and Dashlane do autofill cross-origin iframe, but autofill the password for the domain of the top-most window (i.e., domain displayed in the URL bar), preventing an attacker from stealing the cross-domain credentials.

Chrome, Edge, Internet Explorer, Opera, and Safari all require user interaction before they will autofill passwords into a cross-domain iframe, though this still leaves them vulnerable

to clickjacking attacks. Firefox defaults to not requiring user interaction before autofilling passwords into cross-domain iframes, leaving it vulnerable to the domain harvesting attack by default.

3.4.3 Fill Form Differing from Saved Form

Password managers detect when a user manually enters a password into a login form and will then offer to save that password for later use. When the password manager later fills this password, it can check that the form to be filled is similar to the form used when the password was saved (e.g., same path or protocol). These types of checks help ensure that the user is entering their password in a non-compromised form that has security equivalent to the form they were using when they first saved their password. Still, there are many situations where it makes sense for the form to have changed—for example, the password was saved on a registration form. (i.e., not a login form).

As such, we gave password managers a full-dot if they either disallowed filling the form or showed the user a notification when there was some disparity between the fill form and the form used to save the password. A half-dot was given if the password manager required user interaction when there was a disparity, but only if this user interaction couldn't be disabled (as it can be in Bitwarden and RoboForm). Note that 1Password X and Safari always require user interaction and therefore always receive at least a half-dot. In the results discussed below, we only highlight when password managers act differently due to discrepancies in the login form.

Password managers do not react to discrepancies in the URL the form is served at (other than checking that the domains match). If the password was saved on a form served over HTTPS, Chrome and Opera will refuse to fill it in a form served with a bad HTTPS certificate, with Edge and IE requiring user interaction. If the form is instead served over HTTP, 1Password X and Dashlane will warn users and Chrome, Edge, Firefox, IE, and Opera will refuse to fill the password. Also, LastPass will force user interaction.

If when the page is first loaded there is discrepancy in the form's `action` property (the URL the password will be submitted to), KeePassXC, LastPass, and Firefox will display a warning, with Firefox also refusing to fill the password. If the `action` property is changed

after page load (i.e., dynamically), KeePassXC and Firefox will display a warning, though unlike before Firefox will go ahead and fill the password. Passwords managers do not react to a similar discrepancy in the `method` property. If the `input` fields in the form have been renamed or removed, LastPass will require user interaction.

3.4.4 Non-Standard Login Fields

We investigated whether password managers would fill form fields with `type='text'` (as opposed to `type='password'`), finding that only DashLane would autofill the password in this case. We also examined whether the tools would autofill a minimal form (i.e., a non-login form), containing only two input fields: a text field and a password field; autofilling in this situation reduces the effort required for an attacker to harvest credentials. In this case, we found that Bitwarden, Chrome, Edge, Firefox, IE, and Opera would all autofill these non-login forms, with the remaining browsers only filling them when explicitly requested to by the user.

3.4.5 Potential Mitigation

Stock et al. [60] recommended a more secure form of autofill that would address XSS-vulnerabilities. Instead of filling the password onto the webpage, where it would be vulnerable to XSS attacks, a nonce was filled into the website as the password. When the nonce was about to be transmitted on the wire to the website, the password manager would then replace the nonce with the real password. This approach prevents JavaScript on the webpage from ever being able to access the user's password. Additionally, the password manager can check that the password is being sent only to the website associated with the password and that the password form is not submitting to a different website.

We checked all the password managers to see if they supported this functionality and found that none of them did. In our investigation of this feature, we tried to implement it ourselves and found that browsers did not allow extensions to modify the request body, preventing extension-based password managers from leveraging this more secure mode of

operation.⁷ Enabling secure password entry is an area where browsers could do more to improve authentication on the web and is discussed in greater depth in Section 5.4.

Silver et al. [58] and Stock and Johns [60] also explored whether setting the `autocomplete` attribute to “off” on the password field would prevent password managers from storing or autofilling the password. We found that no password manager obeys this attribute.

Looking at the current W3C specification, it is unclear whether the `autocomplete` attribute should preclude storage and autofill of login credentials [65]. While the specification does state that the “user agent” should not fill fields marked with `autocomplete`, it is unclear if this is only referring to primary user agent (i.e., the browser) or also user agent extensions (i.e., the password manager). Mozilla’s documentation also notes that in order to support password manager functionality, most modern browsers have explicitly chosen to ignore the `autocomplete` attribute for login fields. [45]. This helps explain why no password managers currently obey this parameter, even though in prior research there was some support for this attribute in browsers [58, 60].

3.4.6 Web Vault Security & Bookmarklets

In their analysis of extension-based password managers, Li et al. [39] showed that problems with the security of online password vaults could magnify autofill issues. These web vaults include both standalone interfaces to the password vault as well as acting as the synchronization backend for extension-based password managers. For example, cross-site request forgery (CSRF) could be used to change the URL associated with a set of credentials, allowing all the user’s credentials to be autofilled and stolen from a single malicious domain. Alternatively, XSS vulnerabilities on a web vault could be used to steal all its passwords.

We evaluated the five extension-based password managers and their web vault backends to see if they had properly addressed potential CSRF and XSS attacks. We found that 1Password X, Bitwarden, DashLane, and LastPass use CSRF tokens to prevent CSRF attacks. RoboForm does not appear to use CSRF tokens and we were able to launch a CSRF attack against its web vault that changed the session timeout parameter. We were unable to find

⁷It may be possible to allow extensions to support this functionality in Internet Explorer using its COM-based extensions, though the documentation is unclear in this regard.

other CSRF attacks as the web vault appears to use cryptographic authentication and not cookies to authenticate other requests.

To evaluate the susceptibility of the web vaults to XSS attacks, we manually inspected each web vault’s content security policy (CSP) headers. The results of this evaluation found no issues with either 1Password X or Dashlane’s CSP policies. Bitwarden’s policies had two small issues: `script-src` allows “self” and `object-src` allows “self” and “blob:”. LastPass’s policies allow for “unsafe-inline” in the `script-src`, leaving a significant opening for XSS attacks. RoboForm did not have any CSP policy for their website. We did try to craft XSS exploits for both LastPass and RoboForm, but these efforts were unsuccessful as both sites employed extensive input sanitization; regardless, both web vaults would benefit from implementing stricter (or any) CSP policies.

Finally, we examined whether extension-based password managers still have bookmarklet-based deployment options (used to support mobile devices) that are vulnerable to attack [39]. We found that other than LastPass, the extension-based password managers no longer support a bookmarklet-based deployment. In their place, password managers rely on native mobile applications to handle password management on mobile devices. LastPass’s bookmarklets correctly execute code inside a protected iframe and filter dangerous messages sent to the bookmarklet, addressing the types of problems found by Li et al. [39].

3.5 Discussion

Our research demonstrates that app-based and extension-based password managers are improved compared to how these types of tools performed in prior studies [29, 39, 58, 60]. In general, they have done a good job at addressing specific vulnerabilities: improving the protection of metadata stored in password vaults, removed (insecure) bookmarklets, limited the ability to autofill in iframes (preventing password harvesting attacks), and addressed web security problems in the online password vaults. On the other hand, there has been little change from earlier work in how they handle passwords for areas without specific vulnerabilities: warning users about discrepancies between the fill form and form where the password was saved or implementation of XSS mitigations. Similarly, browsers-based

password managers continue to significantly lag behind app-based and extension-based password managers, both in terms of security and functionality.

Based on our findings, we recommend that users avoid Firefox’s built-in password manager. In particular, its autofill functionality is extremely insecure, and it is vulnerable to a password harvesting attack [58, 60]. If an attacker can mount network injection attacks against a user (e.g., control a WiFi access point), then it is trivial for that attacker to steal all credentials stored in the user’s Firefox password vault. Hopefully, these issues will be addressed when Firefox transitions to their Firefox Lockbox password manager. Users of KeePassXC’s browser extension should also ensure that they do not disable the user interaction requirement before autofill, as doing so will also make the client susceptible to the same password harvesting attack.

We also suggest that users should eschew browser-based password managers in favor of app- and extension-based password managers, as the latter are generally more feature rich, store passwords more securely, and refuse to fill in passwords in a cross-origin iframe. The one exception to this is Safari’s password manager, which does a good job of storing passwords and avoids autofill mistakes, though it does lack a good password generator.

With the app- and extension-based password managers there is still a need for users to ensure that they are properly configured. Neither Dashlane nor LastPass require user interaction before autofilling passwords into websites, and Bitwarden and Roboform allow this interaction to be disabled. If user interaction is disabled, a user that visits a compromised website (e.g., an attacker has exploited an XSS vulnerability) can have their password for that site stolen without the user being aware that this has happened. While this is not as bad as a password harvesting attack [58, 60] (which is now prevented by extension-based password managers), it is still a vulnerability that users should not need to know or worry about. Of the extension-based password managers we studied, only 1Password X refuses to ever autofill passwords.

In the remainder of this section, we describe our recommendations to improve functionality within existing password managers. We also identify several areas for future research that have the potential to significantly improve the utility and security of password managers.

3.5.1 Filter weak passwords.

Our research shows that password managers will randomly generate passwords that can be trivially cracked by online- or offline-guessing attacks. This is a natural extension of password generation being truly random—i.e., any password can be generated, even if it is a natural language word with common substitutions (e.g., “d@rKn3s5”) or exhibits repeated characters patterns (e.g., “'+'+'+_+”). While this is extremely unlikely for passwords of sufficient length (10 characters for online resistance, 18 for offline resistance), it is still possible. To address this problem, we recommend that password generators add a simple filter that checks if the generated password is easily guessable (easily checked using `zxcvbn`), and if so, generate a replacement password.

3.5.2 Better master password policies.

Password managers require that users select and manage a master password, with the hope because they only need one password that users will select a sufficiently strong secret. If users fail to pick a good master password, especially if the selected master password is not online-attack resilient, then a password manager becomes a single point of failure for that user’s accounts. Unfortunately, trusting users to always choose strong master passwords is problematic for three reasons: (1) users don’t necessarily understand what constitutes a strong password, (2) their chosen passwords might have transformations they consider unique but turn out to be common, and (3) users might still select an easy password because it is more convenient.

For these reasons, we recommend that password managers adopt stringent requirements for master password selection, preventing users from turning their password manager into a single point of failure. Additionally, password managers should all transition to using memory hard KDFs for transforming the master password into an encryption key.

3.5.3 Safer autofill.

Autofilling credentials without user interaction puts those credentials at risk if the website is compromised by an XSS attack. For this reason, we recommend that password managers

default to require user interaction before autofilling passwords. Where possible, we also suggest removing the option to disable user interaction as users are unlikely to understand the implications of turning it off. Autofilling into iframes, same- or cross-origin, is also dangerous as it allows clickjacking attacks to circumvent user interaction requirements. As such, we recommend disabling autofill with iframes, or if that is not feasible to consider moving the user interaction out of the web page and into the browser—as Bitwarden and RoboForm do—making clickjacking attacks much more difficult.

Chapter 4

A Security Analysis of Autofill Frameworks on iOS and Android

On desktop environments, password managers are primarily implemented as ad-hoc browser extensions—i.e., the extension individually implements all aspects of the autofill process without support from OS or browser autofill frameworks. While some desktop password managers correctly achieve P1 and P2 [48], many have incorrect implementations that allow attackers to steal or phish users’ credentials [39, 58, 60, 48], and none are able to fully implement P3 due to technical limitations of browser extension APIs [60, 48].

In contrast to the situation on desktop, mobile operating systems provide system-wide autofill frameworks that attempts to standardize and secure the autofill process. Critically, these frameworks have the potential to enforce correct handling of P1–P3 for all mobile password managers. Additionally, these frameworks provide support for autofill within apps, which is largely unavailable on desktop.

In this paper, we conduct the first evaluation of the mobile autofill frameworks, examining whether they achieve substantive benefits over the ad-hoc desktop environment or become a problematic single point of failure. In this evaluation, we consider all such frameworks: iOS’s app extensions, iOS’s Password AutoFill, and Android’s autofill service. Positively, our evaluation finds that all frameworks correctly require user interaction before autofilling credentials (P1), a marked improvement over the mixed support on desktop. In contrast, we find that framework support for P2 and P3 is severely limited.

Within browsers, we find that the framework does not check that the webpage to be filled was sent over a secure channel, nor that filled credentials will be sent to the appropriate domain upon form submission. The framework also fails to provide sufficient information about the webpage to the password managers, preventing the managers from making appropriate security checks themselves. Overall, this leads to mobile password managers being less secure than their desktop counterparts.

Within apps, autofill behavior differs based on the type of interface being filled: (1) native UI elements, (2) WebView controls, and (3) custom-drawn UI elements, of which the first two are supported by mobile autofill frameworks. For native elements, we find that iOS Password AutoFill provides a strong and secure binding between credentials and apps, achieving P2 and P3. In contrast, the Android autofill service provides no such binding, leaving the mapping of credentials and apps to the individual password managers, with nearly all such mappings being insecure (breaking P2). Even worse, iOS app extensions not only fail to provide a secure mapping mechanism but also prevent managers from implementing their own mappings, allowing any credential to be autofilled into any app.

For WebView controls, credentials should only be autofilled if they match the domain of the webpage within the WebView control, regardless of which credentials are mapped to the app. Such behavior is enforced by iOS Password Autofill (achieving P2), whereas both iOS app extensions and the Android autofill service leave the mapping decision to the individual password managers, with only some managers implementing the mapping correctly. Managers that do not properly implement this mapping instead autofill the app’s mapped credentials into the webpage, allowing malicious or compromised webpages displayed in benign apps to phish the app’s mapped credential (breaking P3). Even in the frameworks and managers that do enforce a secure mapping, we identify a limitation in the design of WebView controls on Android and iOS that allows a malicious app to host benign webpages within a (potentially invisible) WebView and steal filled credentials, enabling the surreptitious phishing of all the user’s credentials (also breaking P3).

Critically, in both phishing attacks, the password manager acts as a confused deputy, displaying the autofill dialog and suggesting that the user fills the credential being targeted by the attack. This is extremely problematic, as in all other contexts the autofill dialog is an

indication that phishing is not occurring, and is designed to give users confidence in filling their credentials. As such, users are unlikely to look carefully at the autofill dialog, increasing the probability that their credentials will be successfully stolen.

Overall, our results show that there are significant issues with mobile autofill frameworks. This does not mean that they are the wrong way forward, but rather that they need to be sufficiently improved so that they can live up to their full potential. We conclude this paper by providing recommendations for how to improve autofill frameworks, and also detail how the design of WebView controls could be improved to address the phishing attacks identified in this paper.

4.1 Background

In this section, we first provide general background on password managers. We then provide specific details regarding secure autofill. We finish by providing information about WebView controls.

4.1.1 Password Managers

Password managers serve to help users (a) create random, unique credentials for each service they authenticate to, (b) store the user’s credentials (both generated and user entered), and (c) fill those credentials for the user. These features provide several tangible benefits for users:

1. They reduce the cognitive burden of remembering usernames and passwords. While users must generally still remember a strong master password, the assumption is that this is easier to do than memorizing the tens or hundreds of passwords they would otherwise need to.
2. Due to this reduced cognitive burden, they make it possible for users to have unique passwords for every service they authenticate to, addressing the problem of password reuse.
3. They help users generate passwords, avoiding weak passwords that would be vulnerable to online and offline guessing attacks.

4. They audit user credentials, helping users identify and replace reused or weak passwords, as well as notifying users of password breaches that involve the user’s credentials.

4.1.2 Secure Autofill

Autofilling credentials is a multi-step process. First, the password manager must detect the login form to be filled. Second, it must identify the correct credentials to fill into the login form. Third, the manager must ensure that filling the credentials is safe. Finally, the manager will actually fill the appropriate credentials.

By examining past research [39, 58, 60, 3, 48], we have systematized three properties that need to be guaranteed by password managers in the third step above if autofill is to be secure:

- **P1—User authorization.** Requiring user authorization before filling credentials helps reduce the attack surface by limiting how often credentials are filled and thus vulnerable to theft [39, 58, 48]. Without interaction, credentials would be automatically filled into any and all login forms, leaving them vulnerable to theft if an attacker can control the page’s contents (for example, XSS vulnerabilities remain common [62], supply chain attacks are increasingly regular [71], and network attacks are feasible when users connect to public WiFi access points [47]). While P3—if properly enforced—protects against such attacks, leveraging P1 as well provides defense-in-depth. This defense-in-depth is especially important as both the current paper and past work [39, 58, 60, 48] demonstrate that P3 is poorly supported in most contexts.

Note, it should not be assumed that users will carefully examine these dialogs, as they likely will become habituated to clicking through them [19, 24]. Instead, requiring interaction is primarily intended to prevent the surreptitious entry and subsequent theft of credentials—for example, Firefox’s built-in password manager fails to require user interaction and due to other flaws in its implementation allows an adversary to silently steal all of the user’s credentials [48]; such an attack would be difficult if not impossible to conduct if user interaction was required as the attack would trigger thousands of autofill requests, alerting the user that something was wrong [48]. Similarly, requiring interaction can also help prevent attacks when the user is not trying to authenticate,

in which case they are more likely to click out of the autofill interface to resume their usage of the the webpage or app, thus preventing potential credential theft.

- **P2—Secure credential to destination mapping.** Managers need to be able to map credentials to the webpage and apps they should be filled in, preventing other webpages and apps from accessing those credentials [3]. Such a mapping prevents malicious webpages and apps from stealing credentials intended for other webpages and apps—i.e., phishing attacks. Most commonly, this mapping is done by associating credentials with domains, then associating those domains with apps. *When properly implemented, the autofill interface then becomes a sign to users that they are not being phished and can safely enter the suggested credentials on the current site or app.*
- **P3—Credentials are only accessible to mapped destinations.** Managers should ensure that after credentials are filled, that they will only be accessible to mapped destinations [60, 48]. For webpages, that means that the credentials will only be sent to a server on the same domain and that they will not be accessible to malicious JavaScript running on the page that may try to exfiltrate the credentials to different domains [60]. For apps, this means that credentials should not be available to other apps on the system. Finally, it also means that if the app is hosting a webpage in a WebView control, that the webpage cannot access credentials intended for the app, nor vice-versa.

4.1.3 WebView

WebView controls are UI widgets provided by the mobile operating system that serve as embeddable, minimalist browsers. They are used by apps to display web content directly within the app, as opposed to having users click a link that opens the main mobile browser. On Android, WebViews are added using the `WebView`¹ class, whereas on iOS they are added using the `WKWebView`² class. In both cases, the WebView control is implemented using WebKit.

¹<https://developer.android.com/guide/webapps/webview>

²<https://developer.apple.com/documentation/webkit/wkwebview>

To allow for integration between a hosting app and the content in the WebView, both iOS and Android allow the WebView control to be styled by the hosting app. This styling is arbitrary and can even go so far as to make it impossible to distinguish between native widgets and content displayed in the WebView. As such, it is possible for the app to prevent the user from knowing they are interacting with content from a domain the app should not have access, making phishing attacks trivial [40].

Additionally, the hosting app is able to inject JavaScript into the content hosted in the WebView. While this is intended to enable bidirectional communication between the app and the WebView content, in practice it allows the app arbitrary control of the WebView content. For security reasons, the Android developer documentation recommends that WebView only be used to show trusted, *first-party* content. However, prior work by Yang et al. [70] found that many popular apps—such as Google News, Facebook, and Uber—load third-party, untrusted content in a WebView and that 11K of the 17K most popular free apps on Google Play contained entry points to WebView loading APIs.

4.2 Evaluation Methodology

There are three mobile autofill frameworks available on iOS and Android (the dominant mobile platforms). On iOS, there is the app extensions framework and the Password AutoFill framework. On Android, there is the autofill service. Prior to the availability of the Android autofill service, many password managers used the Android accessibility (a11y) service to hack in support for autofill. We chose not to include the a11y service in our evaluation for two reasons: first, it is not and designed as an autofill framework and thus doesn't serve as a meaningful point of comparison and second, it is now well-known that the accessibility service is ill-suited to be used for security purposes [36, 38, 28, 46].

As part of our evaluation, we identified three contexts in which autofill occurs on mobile devices:

1. Webpages displayed in mobile browsers.
2. Native UI elements (i.e., widgets provided by the OS) presented within mobile apps.

3. Webpages displayed in WebView controls presented within mobile apps.

In our evaluation, we consider all three of these contexts. While the requirements to satisfy P1 is the same for each of these contexts, they divergent for P2 and P3. As such, we used different tests in each context to explore the security of autofill. These context-dependent tests along with their results are given in §4.3, §4.4, and §4.5, respectively.

In the remainder of this section we give more details about the three mobile autofill frameworks we tested. We then give details about our overall testing approach.

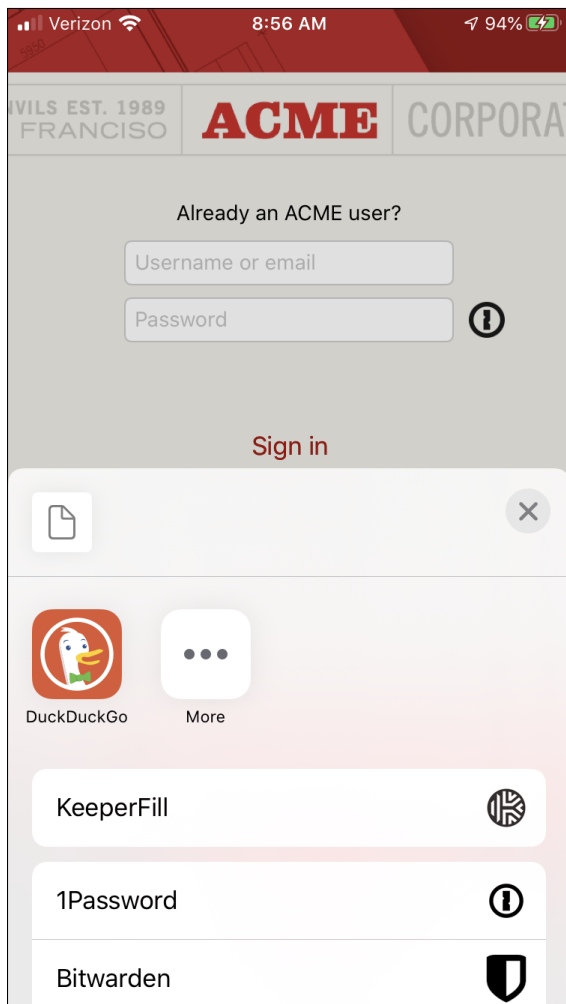
4.2.1 Mobile Autofill Frameworks

Below we provide background on the three mobile frameworks tested in our work. For those interested, Appendix A provides a historical overview of credential filling on iOS and Android.

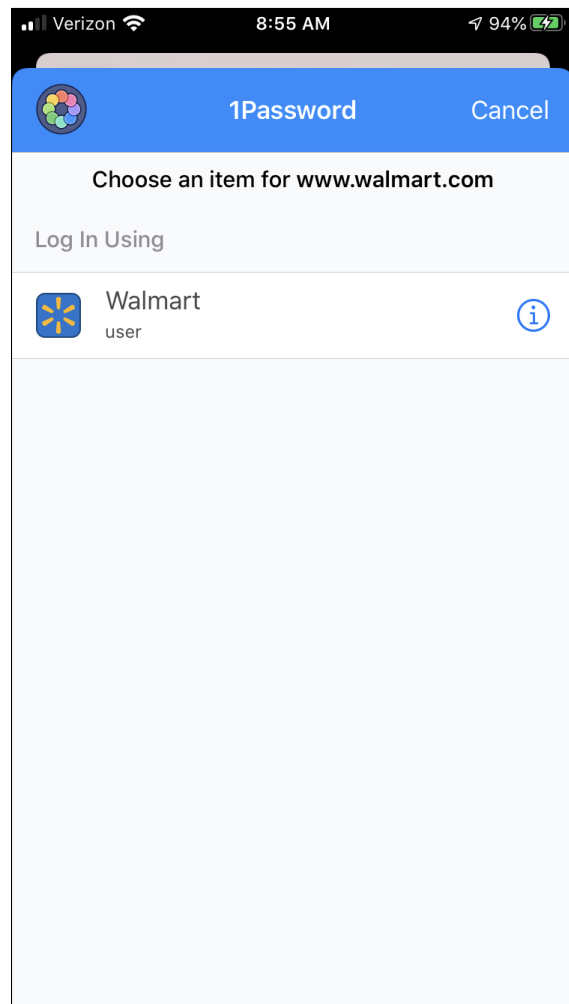
iOS App Extensions

App extensions were introduced in iOS 8 (2014) and allow a host app to interact with another app (e.g., a password manager) using a predefined set of extension features. For password autofill, this requires the password manager to implement the set of functions associated with the password management extension feature and for host apps to be updated to query this extension feature. Note, the functionality provided by app extensions is minimal, enabling autofill, but not attempting to secure it. For example, host apps are trusted to identify which credentials they should receive, with the framework doing nothing to check this mapping or verifying that the host app is not sending those credentials to a different domain. Figure 4.1 shows the interface for app extensions, first requiring the user to select which app extension to use (see Figure 4.1a) and then selecting the credentials (see Figure 4.1b).

While superseded by iOS Password AutoFill, we included iOS app extensions in our evaluation for three reasons: (1) it is still supported in iOS and remains functional in some password managers (e.g., 1Password, Keeper, LastPass) and host apps (e.g., Safari, Edge); (2) for older devices that cannot be updated to iOS 12, app extensions remain the preferred method for password autofill; (3) it provides a distinct approach to designing frameworks which provides a helpful point of comparison to the other two frameworks.



(a) Selecting app extension



(b) Selecting password

Figure 4.1: iOS App Extensions UI

iOS Password AutoFill

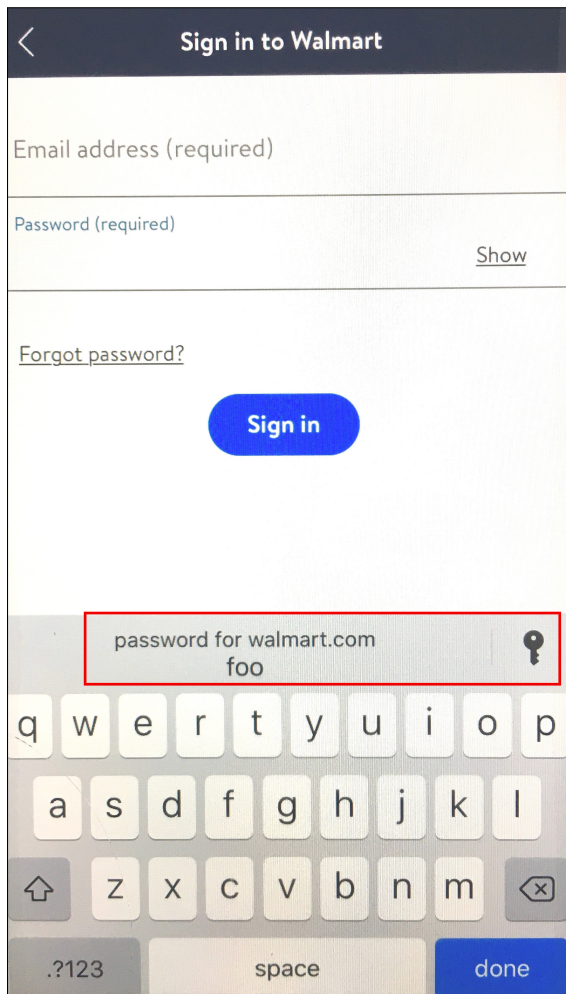
The Password AutoFill framework was introduced in iOS 12 (2018) and takes a radically different approach to autofill. Whereas app extensions provided minimal functionality, Password AutoFill controls the entire autofill process in an attempt to improve the usability and security of password autofill. First, it handles the identification of login forms in both apps and websites, though the host app can help this process by annotating appropriate fields using the `textContentType` attribute. Second, Password AutoFill ensures a secure mapping between an app and the domains that should have their credentials entered into that app. That is done by having app developers include an Associated Domains Entitlement that indicates which domains are associated with the app; the domain operator is also required to include an `apple-app-site-association` file on their website indicating which apps are allowed to use credentials for that domain. Third, Password AutoFill handles both the UI shown to users and the actual entering of credentials into the target app. Figure 4.2 shows the interface for Password AutoFill, both when an associated domain can be found (see Figure 4.2a) and when not (see Figure 4.2b).

Android Autofill Service

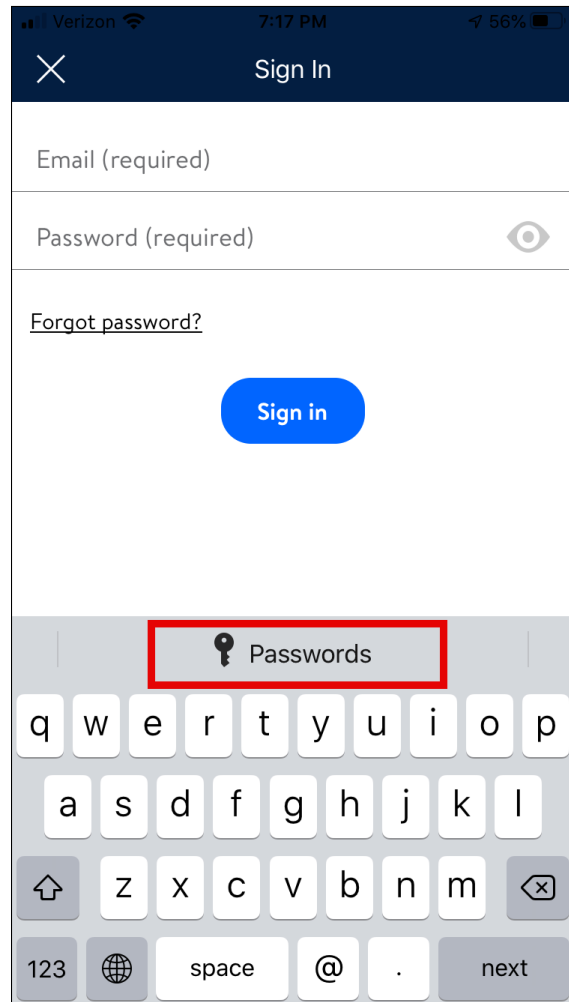
In 2017, Android introduced the autofill service as part of API 26 (Android 8.0—Oreo). This service falls between iOS’s app extensions and Password AutoFill in terms of the features it supports. Like Password AutoFill, it handles the identification and filling of login forms for apps and websites, with apps being able to help this process by annotating interfaces using the `android:autofillHints` attribute. Unlike Password AutoFill, it does not control the credential selection UI (Figure 4.3 gives two examples of UIs provided by password managers on Android), nor does it enforce any app-to-domain credential mapping.

4.2.2 Testing Approach

Our evaluation of the three mobile autofill frameworks was primarily empirical in nature. More specifically, we selected and evaluated 14 mobile password managers, each of which was implemented using one or more of the frameworks under test. These managers were chosen



(a) Credential found



(b) No credential found

Figure 4.2: iOS Password AutoFill UI

as they are the most popular password managers implement with the frameworks under test. Table 4.1 summarizes these password managers, which frameworks they support, and which versions were tested. Download counts for each tool are given in Appendix B.

For each of these managers, we conducted a series of test designed to evaluate how well the manager enforced P1–P3 (see §4.1.2). Within these evaluations, we payed attention to when the results were either the same and when they were different, helping us measure to what extent P1–P3 were enforced by the framework, to what extent the frameworks left implementing these properties to the individual managers, and to what extent the frameworks limited the ability of the individual managers to achieve these properties. In addition to this empirical evaluation, we also reviewed the documentation and APIs for each framework to try an contextualize and confirm our results. We also reached out to developers of the password managers to understand the results, though only a few responded to our requests for information.

Testing in iOS was performed using an iPhone 7 running iOS 13. Testing on Android was conducted using the Genymotion Android Emulator to simulate a Google Pixel 2 device running Android 9 (Pie).

4.3 Browser Autofill

We begin our investigation by exploring autofill within mobile browsers. As the security concerns for autofill on mobile browsers are the same as those on desktops, we base our methodology on the recent evaluation of desktop password managers conducted by Oesch and Ruoti [48]. We choose to use this methodology for two reasons—first, it is complete, covering all necessary aspects of browser autofill security and second, it allows us to directly compare the performance of mobile password managers to desktop managers. For each of the tests described by Oesch and Ruoti, we identify which of the three properties we synthesized (P1–P3) that the test relates to, removing extraneous tests that identify interesting edge cases but which cover features not actually needed to implement autofill securely.³ We also

³We contacted Oesch and Ruoti regarding our removal of tests and they agreed that they were appropriate.

Table 4.1: Analyzed password managers on iOS and Android

System	iOS Password AutoFill iOS app extensions Android autofill service			iOS	Android
	Framework	Version			
1Password	✓ ✓ ✓	7.4.7	7.4		
Avast Passwords	✓ ✓ ✓	1.15.4	1.6.4		
Bitwarden	✓ ✓ ✓	2.3.1	2.2.8		
Dashlane	✓ ✓	6.2013.0	6.2006.3		
Enpass	✓ ✓ ✓	6.4.2	6.4.0		
iCloud Keychain	✓	13.3.1	—		
Keepass2Android	✓	—	1.07b-r0		
Keeper	✓ ✓ ✓	14.9.1	14.5.20		
LastPass	✓ ✓ ✓	4.8.0	4.11.4		
Norton	✓ ✓ ✓	6.8.78	6.5.2		
RoboForm	✓ ✓ ✓	8.9.2	8.10.4		
SafeInCloud	✓ ✓	20.0.1	20.2.1		
Smart Lock	✓	—	9.0		
StrongBox	✓	1.47.4	—		

considered if there were any additional tests needed to satisfy the three properties, but found that the trimmed set of tests was sufficient to fully evaluate P1–P3.

All tests were performed using the default browser for each operating system: Safari (v13.3.1) for iOS and Chrome (v 81.0.4044) for Android. We chose to focus on the default browser as they are likely the most widely used browser on each respective platform. Additionally, the browsers are developed by the same company developing the autofill framework, allowing us to measure the security of the frameworks at their best.

In the remainder of this section, we describe the paired down tests along with our results. These results for each framework are summarized in Table 4.2. To allow for an easy comparison with desktop managers, Table 4.2 also provides the ratings for the most and least secure desktop managers from the prior work [48]. The full results for individual managers can be seen in Appendix C.

4.3.1 P1—User Interaction

We tested whether this property was supported by constructing a login webpage and visiting it in the mobile browser, recording whether user interaction was required before credentials were filled. We visited this webpage over HTTP, HTTPS, and using HTTPS with an invalid certificate to test whether this impacted user interaction requirements (as it does on some desktop browsers).

Our results show that all three frameworks correctly require user interaction before filling credentials. This is a marked improvement over desktop managers, where only 2 of the 12 managers tested by Oesch and Ruoti enforced user interaction. This result shows the potential for autofill frameworks to enforce correct behavior across all password managers.

Finding #1: Within browsers, P1 is enforced by all frameworks, a marked improvement over the situation on desktops.

Table 4.2: Autofill Security in Mobile Browsers

Framework	P1	P2	P3
iOS Password AutoFill	●	● ○ ○	○ ○ ○ ○ ○
iOS App Extensions	●	● ○ ○	○ ○ ○ ○ ●
Android Autofill Service	●	● ○ ○	○ ○ ○ ○ 
Most secure desktop manager [48]	●	● ◐ ●	○ ◐ ◐ ◐ ●
Least secure desktop manager [48]	○	● ● ○	○ ● ● ○ ○

- Secure behavior ◐ Partially secure behavior
 ○ Insecure behavior  Delegated to password manager

4.3.2 P2—Credential-to-Domain Mapping

To test credential mapping, we first registered credentials for different domains. We then created testing webpages across multiple domains and checked that each domain only received credentials that were appropriate for that domain.

We also tested if the credential mapping took into account whether the source of the webpage was authenticated using HTTPS. To do this, we created test pages that were served over HTTP and over HTTPS with an invalid certificate, respectively, observing whether autofill would proceed or not. Ideally, autofill would not be allowed in these cases, as such occurrences could represent a network attack being used to steal credentials.⁴ We also allow for a rating of partially secure when autofill is allowed, but only after notifying the user of the potential danger.

We find that while all password managers do map credentials to their appropriate domains, none of them check whether the webpage was served over a secure HTTPS connection, thus leaving open the possibility of network injection attacks. Worse yet, the frameworks provide insufficient information to the password managers, preventing them from checking and enforcing this property themselves. This leads to mobile managers being as bad as or worse than even the least secure desktop managers in regards to P2.

Finding #2: Within browsers, P2 is only partially enforced by the frameworks. The frameworks also prevent the managers from being able to enforce this property, causing mobile managers to be less secure than all desktop managers.

4.3.3 P3—Protecting Filled Credentials

The best way to achieve P3 is to only fill credentials into the web request itself, a proposal first described by Stock and Johns [60]. As the credentials are set inside the web request, they are never stored on the webpage and thus cannot be accessed by JavaScript (malicious or benign) executing on the webpage. On desktops, implementing this proposal is not possible as browser extensions lack the permissions to modify the web request; in contrast, this is not

⁴The user can always override this behavior by manually copying and pasting credentials.

a significant limitation on mobile, where both the framework and browser are maintained by the same entity, allowing them to implement advanced protections such as these. We test whether this proposal is used by creating a page with JavaScript that attempts to exfiltrate filled passwords.

When the above approach is not used, it is not feasible to prevent malicious JavaScript from accessing the field credentials. Still, it is possible to limit the likelihood that credentials will be accidentally sent to an unmapped domain. First, the login form’s **action** attribute should be checked to ensure that credentials will be submitted to the appropriate domain. We test this by creating two webpages, one that has the form’s action set to a different domain at page load (our static test), and one that sets the action field after page load, but before the credentials are autofilled (our dynamic test). Second, the login form’s **method** attribute should be checked to ensure that the credentials are not included in the URL (i.g., using a **GET** request), as this could potentially leak the credentials to another domain through the **HTTP Referer** header. We test this by creating a webpage with the method set to **GET** and observing whether autofill is allowed. For both of these checks, we consider secure behavior refusing to fill the credential, with partially secure being awarded if the credential is filled only after warning the user about potential dangers.

Finally, autofill within cross-domain iframes should be disabled [58, 60, 48]. If a user visits a malicious or compromised webpage, the adversary can open a cross-origin iframe to any domain where they can inject JavaScript and steal credentials when those credentials are autofilled. In the worst case, if user interaction is not required, the credential theft will always succeed and will be unnoticed by the user. Even if user interaction is required, the adversary is still able to effectively launch a phishing attack, which is likely to succeed as the appearance of an autofill dialog is supposed to indicate to the user that a phishing attack is not occurring (see §4.1.2). We test this by creating a webpage with a cross-origin iframe and observe whether the iframe triggers autofill.

Our results show that none of the password managers adopt Stock and Johns’ proposal, leaving filled credentials at risk of theft by JavaScript-based attacks. Additionally, none of them check the **action** or **method** fields on the login form. As with P2, they also fail to provide sufficient information to the managers to allow the managers to implement these

features themselves. In regard to cross-origin iFrames, iOS Password AutoFill does not prevent cross-origin autofill, and does not allow any of the managers to override this behavior. In contrast, the older iOS app extensions properly prevent cross-origin autofill. On Android, the autofill service does not prevent cross-origin autofill, but does allow managers to override this behavior. As with P2, the mobile frameworks perform as bad as or worse than even the least secure desktop manager in regards to P3.

Finding #3: Within browsers, P3 is not enforced, leaving filled credentials vulnerable to both theft and accidental leakage. The frameworks also prevent the managers from being able to enforce this property, causing mobile managers to be less secure than nearly all desktop managers.

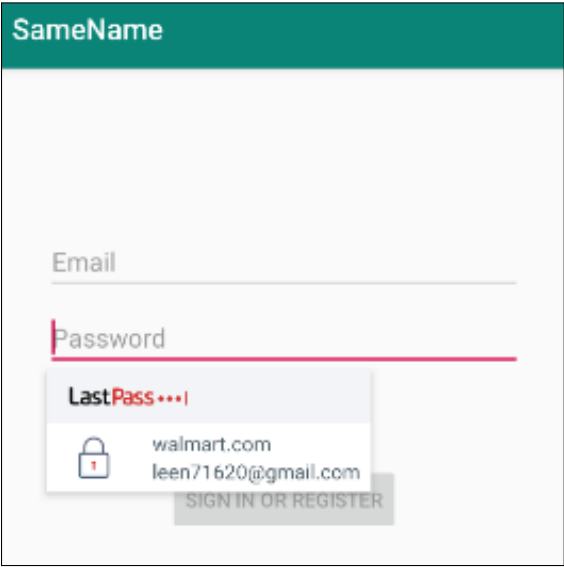
4.4 App Autofill

We continue our evaluation by examining autofill within apps. One of the greatest benefits of the mobile autofill frameworks is that they support autofill within apps, something that is largely unavailable in the desktop environment. There are three different ways login interfaces can be displayed within apps: (1) using native UI elements (i.e., OS-provided widgets), (2) using custom UI elements drawn and managed by the app, and (3) within a webpage displayed in a WebView hosted by the app. The first of these approaches is discussed in this section, while the third is discussed in the next section. The second approach cannot be detected or autofilled by mobile frameworks, and so it is excluded from our analysis.

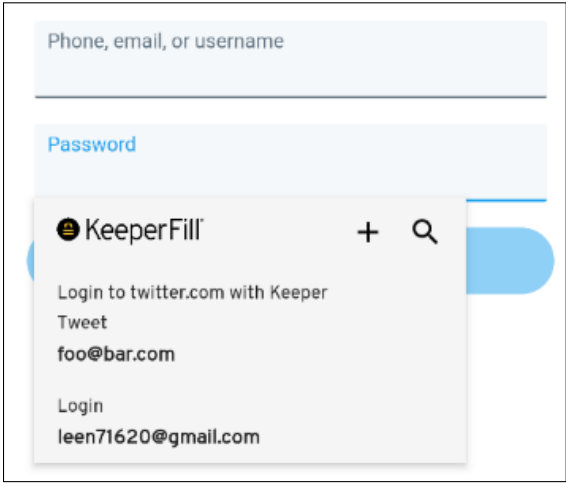
An overview of the results of our analysis is given in Table 4.3.

4.4.1 P1—User Interaction

We tested whether this property was supported by constructing a custom app and attempting to autofill it, recording whether user interaction was required before credentials were filled. Our results show that all three framework correctly require user interaction before filling credentials.



(a) LastPass



(b) Keeper

Figure 4.3: Android Password AutoFill UI

Table 4.3: Autofill Security for Native UI Elements in Apps

Framework	User interaction always required Secure app-to-domain mapping Secure domain-to-app mapping Prevents access from other apps Prevents access from WebView			
	P1	P2	P3	P4
iOS Password AutoFill	●	● ●	● ●	
iOS App Extensions	●	○ ○	● ●	
Android Autofill Service	●	✍ ✍	● ●	

● Secure behavior ○ Insecure behavior

✍ Delegated to password manager

Cross-origin iframe overlay
of any walmart.com page
with XSS vulnerability

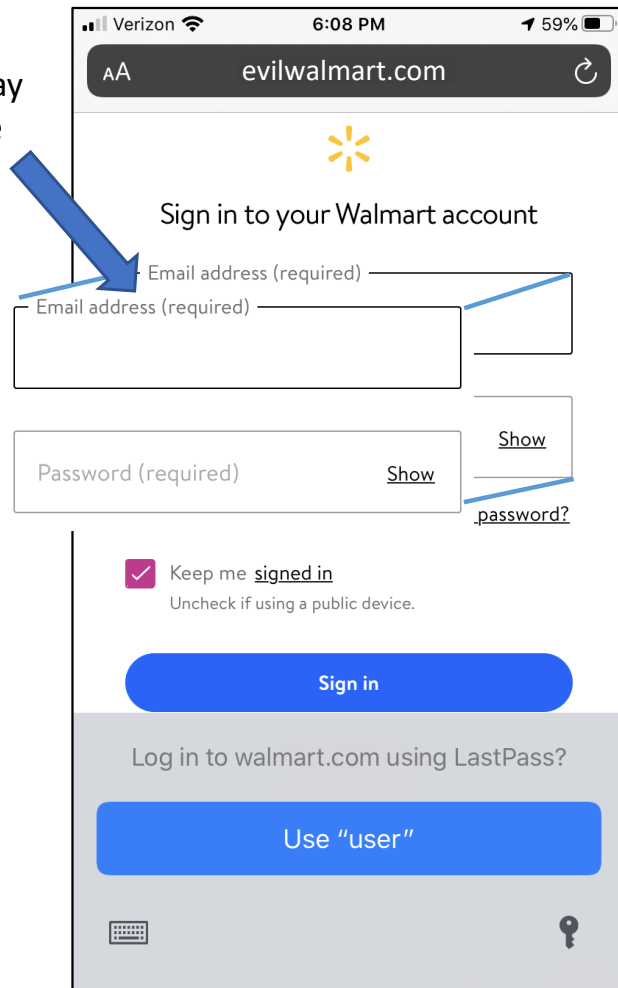


Figure 4.4: Example of a cross-origin iframe phishing attack

Finding #4: Within native UI elements, P1 is enforced by all frameworks.

4.4.2 P2—Credential-to-App Mapping

There are two ways that credentials could be mapped to apps. First, there could be a direct mapping between credentials and apps. Second, there could be a mapping between Web domains and apps, leveraging the existing credential-to-domain mapping to providing an indirect credential-to-app mapping. It is this second approach that is used by all mobile frameworks and managers. This likely arises due to the fact that browser-based autofill significantly predates app-based autofill.

When establishing a domain-to-app mapping, it is ideal if this mapping is bi-directional. First, the app should identify which domains it is associated with, limiting the number of credentials it could access if the app became compromised (an application of the principle of least privilege). Second, a domain should list which apps are allowed access to its associated credentials, preventing a malicious app from being able to access arbitrary domains' credentials. Only when both of these mappings agree should a credential for the given domain be suggested by the autofill framework. Both of these mappings should be secured cryptographically—for example, by (1) code signing the app (including the file identifying the mapped domains), (2) using the fingerprint of the code signing certificate in the identification of apps on the domain's side, and (3) transmitting the domain's mappings using TLS. Note, this bi-directional mapping does impede the use of single sign-on (SSO)—for example, Google's domain is unlikely to whitelist all the apps that use it for SSO—but this use case can be handled by hosting the SSO interface in a WebView (see §4.5).

To evaluate whether the frameworks satisfy these requirements, we first examined the documentation for each framework to understand what process they claimed to use and to identify the mechanisms they used for app-to-domain mappings. This led us to realize that all three frameworks take a drastically different approach to Pp2, each requiring its own testing strategy.

iOS Password AutoFill

Apps indicate the domains they are associated with by including an Associated Domains Entitlement file in their app package. Since this file is part of the app package, its contents are signed as part of the code signing process required for all iOS apps. Domains indicate the apps they are associated with by including an `apple-app-site-association` file at a specific URL on that domain. This file indicates which apps are allowed to use credentials for that domain, with the appropriate code signing key for the each app also being identified. Credentials will only be autofilled if both of these mappings exist for a given domain.

To confirm that this functionality was working as intended, we created several testing apps and domains. These apps included apps with the appropriate mappings and those without. We also created look-alike apps that had all the same information as a legitimate app, but which were not signed with the correct code signing key.

Throughout all of our testing, iOS Password AutoFill performed exactly as it should, providing a secure credential-to-app mapping and satisfying P2.

iOS App Extensions

The iOS app extension framework does not provide a mapping between apps and domains. Instead, it allows the app to arbitrarily dictate which domain's credentials it would like to have autofilled. This allows malicious apps to phish users' credentials, putting the onus on the user to detect that the malicious app should not receive the credentials suggested by the password manager, which as previously discussed runs counter to how autofill dialogs are supposed to work (see §4.1.2). We verified this behavior by building an app that uses app extensions to autofill a login form and verifying that we could request credentials for arbitrary domains.

Android Autofill Service

The Android autofill service does not provide a mapping between apps and domains, instead leaving this functionality up to the individual managers to implement. To analyze the mappings used by the 12 Android password managers we tested, we first inspected the autofill

ceremony to see if domain-appropriate passwords were being suggested. If they were, we used jadyx⁵ to decompile the password manager’s apk file and try to reverse engineer the credential mapping. As part of this effort, we also used appmon⁶ to intercept API calls from the password manager.

In our analysis, we identified four domain-to-app mappings used by the various managers: (a) a static list of app-to-domain mappings; (b) a custom heuristic that matches apps to domains based on the app’s `applicationId` (e.g., `com.google.photos`); (c) digital Asset Links (DAL) files hosted by domains at a specific URL are used to specify which apps, identified using their code signing key, should be mapped to that domain; and (d) relying on manual mappings provided by end users. The list of these mappings along with their satisfaction of P2 is given in Table 4.4, with further details available in Appendix D.

None of these managers use a bidirectional app-to-domain mapping. Only one mapping (DAL) requires domains to identify the apps they are associated with. Only two managers check a cryptographic attestation of app identity, meaning that look-alike and side-loaded apps can impersonate legitimate apps and receive their associated credentials. As such, our results demonstrate that while the autofill service’s decision to delegating mappings could work in theory, in practice it turns out to be a poor design decision.

Finding #5: For native UI elements, iOS AutoFill correctly provides P2. In stark contrast, iOS app extensions provide no credential-to-app mapping, allowing any app to request credentials for domains. The Android autofill service leaves P2 up to managers to implement, but this turns out to be a bad decision with no manager correctly implementing such mappings.

4.4.3 P3—Protecting Filled Credentials

Filled credentials should only be accessible to the app receiving those credentials, not to other apps or by webpages hosted in WebView controls within the filled app. For all three frameworks, this property is satisfied by strong app segmentation guarantees provided by

⁵<https://github.com/skylot/jadx>

⁶<https://github.com/dpnishant/appmon>

each framework’s respective operating system. Note, these protections can be side-stepped on Android using the accessibility service, but this is necessary to support individuals with disabilities, and as such users need to remain careful about which apps they give permissions to control this service.

Finding #6: Within native UI elements, P3 is enforced by the mobile operating system.

4.5 WebView Autofill

We end our investigation by considering autofill within WebView controls hosted inside apps. Such functionality is critical to enable a range of use cases:

1. **Supporting single sign-on (SSO).** As described in §4.4, autofill for native UI elements should only be allowed if the domain owner indicates the app is associated with the domain. As SSO providers are unlikely to whitelist every app that wants to use SSO, apps can instead use an embedded WebView control to display the SSO flow.
2. **Thin-wrapper apps.** Many mobile apps serve as little more than a thin-wrapper around an existing website, with the app displaying a WebView control that displays the wrapped website.
3. **Avoiding duplicating code.** Instead of having one authentication codebase for use on a website and one for the app associated with the website, app developers may instead choose to have authentication handled by the website using a WebView control.

In each of these cases, it is important to ensure that credentials are filled safely. This is especially important in the case of SSO, as those credentials if stolen would have a large impact on the user’s life. An overview of our analysis of autofill security in WebView controls is given in Table 4.5.

Table 4.4: App-to-Domain Mappings on Android

System	Mapping				P2	
	Static	Heuristic	Digital Asset Links (DAL)	Manual association	Secure app-to-domain mapping	Secure domain-to-app mapping
1Password			✓		○	○
Avast Passwords	✓	✓			○	○
Bitwarden		✓			○	○
Dashlane	✓				●	○
Enpass			✓		○	○
Keepass2Android			✓		○	○
Keeper		✓	✓		○	○
LastPass	✓		✓		○	○
Norton	✓				○	○
RoboForm			✓		○	○
SafeInCloud		✓			○	○
Smart Lock			✓		○	●

✓ Supports this functionality ● Secure behavior ○ Insecure behavior

Table 4.5: Autofill Security for WebView Controls

Framework	P1	P2			P3					
iOS Password AutoFill	●	●	○	○	○	○	○	○	○	○
iOS App Extensions	●	🔑	○	○	○	○	○	○	○	●
Android Autofill Service	●	🔑	○	○	○	○	○	○	○	🔑

● Secure behavior ○ Insecure behavior 🔑 Delegated to password manager

Highlighted columns refer to tests unique to or modified for WebView controls.
The remaining columns mirror the tests for mobile browsers.

4.5.1 P1—User Interaction

We tested whether this property was supported by constructing a custom app with an embedded WebView control and attempting to autofill the WebView content, recording whether user interaction was required before credentials were filled. Our results show that all three frameworks correctly require user interaction before filling credentials.

Finding #7: Within native UI elements, P1 is enforced by all frameworks.

4.5.2 P2—Credential-to-Domain Mapping

For WebView, credentials should be mapped based on the domain of the content displayed inside the WebView. To test this, we created a test app mapped to one set of credentials, with the test app including a WebView with content from a domain associated with a different set of credentials. We then tried to autofill a login form within the WebView and examined the set of credentials. In addition to this test, we also replicate the two connection security tests used to evaluate mobile browser autofill.

Our results find that that iOS Password AutoFill correctly maps credentials to the domain displayed in the WebView control. iOS app extensions and the Android autofill service leave the mapping up to individual managers, with only a minority of using the correct mapping scheme.⁷ In contrast, the majority of managers instead autofill credentials into the WebView based on the credentials.⁸ This leaves the app credentials vulnerable to phishing from compromised or malicious content displayed in the WebView, which we describe further below. The remaining managers refuse to autofill credentials into WebView controls, a significant limitation on utility.⁹

For the connection security tests, our results match the poor results for autofill within mobile browsers.

⁷iOS app extensions—1Password, Android autofill service—Dashlane, Keeper, Lockwise, SafeInCloud, and Smart Lock

⁸iOS app extensions—Keeper, Bitwarden, LastPass, and Enpass, Android autofill service—1Password, Bitwarden, Enpass, Keepass2Android, Keeper, LastPass, and RoboForm.

⁹iOS app extensions—Avast, Norton, and Roboform, Android autofill service—Avast.

Phishing Attack #1: Using a Malicious Webpage

In this phishing attack, a benign app uses a WebView to display content from a compromised domain—either because the attacker compromised a domain normally used by the app (e.g., XSS attacks or supply chain attacks) or because the attacker used a different vulnerability to get the app to load a domain of the attackers choosing. Once displayed, the malicious domain displays a (possibly hidden) login form, triggering the autofill ceremony and causing the password manager to suggest the user autofill the app’s associated credentials (as opposed to the domain’s associated credentials) into the WebView. As the autofill dialog is intended to give confidence to users that it is safe to enter credentials (see §4.1.2), it is unlikely that they will even consider the possibility that a phishing attack is occurring. Moreover, depending on the styling of the WebView and the content displayed therein, there may be no visual indication that the user is interacting with content not native to the app (see §4.1.3), eliminating nearly any chance that the user could detect a phishing attack.

To validate this attack, we developed a proof of concept app that displays content from a different domain in a WebView. The displayed domain was considered to be under attacker control and displayed a login form, styling the form to look as if it was part of the hosting app. When the autofill framework suggested credentials, it was the app’s, not the domain’s associated credentials. After clicking through the dialog, these credentials were sent to the malicious domain, successfully completing the phishing attack. Based on our own experience with this proof-of-concept app and prior work [19, 24, 40, 64], we believe it should not be too difficult for adversaries to leverage this phishing attack in vulnerable managers.

Finding #8: Within WebView controls, only iOS Password AutoFill correctly maps credentials to the domain of the content in the WebView. The remaining frameworks leave this mapping up to the managers, with the majority of managers using an incorrect mapping. This incorrect mapping leaves users vulnerable to a phishing attack where benign app credentials can be stolen by compromised content displayed in the WebView.

4.5.3 P3—Protecting Filled Credentials

After filling credentials into the WebView, it should not be possible for the hosting app to extract those credentials. Without this protection, it would be possible for a malicious app to host arbitrary login forms from various domains to steal those credentials. To test this, we created a demo app mapped to one set of credentials, with the demo app including a WebView with content from a domain associated with a different set of credentials. After autofilling credentials, we then attempted to use the hosting app to steal the credentials from the WebView. In addition to this test, we also replicate the five P3 tests used to evaluate mobile browser autofill.

We find that in each case, the WebView allowed the injection of malicious JavaScript into the WebView by the hosting app, with this JavaScript able to find and exfiltrate filled credentials. This leaves all of a user’s credentials vulnerable to phishing by a malicious app, an attack describe more in-depth below.

For the remaining tests, our results match the poor results for autofill within mobile browsers. Note, that if passwords were only filled on transmission, this would have addressed the phishing attack described below.

Phishing Attack #2: Using a Malicious App

On both Android and iOS, the WebView control allows apps to inject JavaScript into any webpage displayed in a WebView. A malicious app can use this feature to inject JavaScript that waits for credentials to be filled into the WebView and then exfiltrate them back to the malicious app. After injecting the appropriate code, the malicious app triggers the autofill dialog by selecting the login elements hosted within the WebView, causing the autofill framework to suggest the user autofill the domains’s associated credentials. The adversary is also free to style the app, the WebView, and the content hosted in the WebView so that it is impossible for the user to tell that they are interacting with a WebView—for example, styling the WebView so that only a single text entry box (e.g., the username field) is shown, with no indication that the textbox it is not a native UI element.

As the autofill dialog is intended to give confidence to users that it is safe to enter credentials (see §4.1.2), it is unlikely that they will even consider the possibility that a phishing attack is occurring. Still, if the adversary were to immediately try to steal all the user’s credentials, this would likely be detected as the user would be bombarded with hundred of autofill dialogs. Instead, a careful adversary could stagger phishing attacks to instances when the user expects to authenticate within the app, stealing credentials over a long period of time.

We developed proof of concept apps for Android and iOS that implement this attack. This apps is styled to look like the Walmart app and phishes credentials when users would attempt to login. Listing 4.1 gives the critical code used to implement the attack. In both apps, we were able to confirm that our app was able to trigger autofill requests for arbitrary domains. Additionally, based on our experience with this app and past research into phishing [19, 24, 40, 64], we believe it is unlikely that users would detect the attack.

Finding #9: Within WebView, P3 is not enforced, leaving filled credentials vulnerable to malicious apps and other Web vulnerabilities.

4.6 Password Generation

In addition to evaluating autofill on mobile devices, we did conduct a less extensive analysis of generation on iOS and Android. All but three of the password managers we evaluated (Lockwise, Firefox, Edge) supported generation, though aWallet only offers this feature in the paid version of their app. Three (iCloud Keychain, MyPasswords, Chrome) of the password managers offered no configuration options in their password generator, while other password managers had either basic configuration options (Avast, Keeper, LastPass, Norton) or more advanced configuration options (1Password, Bitwarden, Dashlane, Enpass, RoboForm, SafeInCloud, StrongGBox, aWallet, Password Safe). Figure 4.6 demonstrates the difference between basic and advanced configuration.

Table 4.6 summarizes the options provided by each password manager’s generator. Several of the password managers we analyzed (Avast, Bitwarden, Dashlane, Norton, SafeInCloud,

```

1 let controller = WKUserContentController()
2 controller.add(self, name: "callbackHandler")
3
4 func userContentController(_controller: WKUserContentController, didReceive
   message: WKScriptMessage) {
5     if(message.name == "callbackHandler") {
6         print("User credentials are \(message.body)")
7     }
8 }

```

(a) Malicious iOS app

```

1 var username = document.getElementById("email").value;
2 var password = document.getElementById("password").value;
3 var credentials = `window.location.hostname:username:password`;
4 window.webkit.messageHandlers.callbackHandler.postMessage(credentials);

```

(b) Injected JavaScript for iOS WebView

```

1 public class WebAppInterface {
2     Context ctx;
3     WebAppInterface(Context c) {ctx = c; }
4
5     @JavascriptInterface
6     public void stealCredential(String domain, String uname, String pword) {
7         Toast.makeText(ctx, String.format(s:%s:%s", domain, uname,
8             pword),Toast.LENGTH_SHORT).show();
9     }
10 }

```

(c) Malicious Android app

```

1 var uname = document.getElementById("email");
2 var pword = document.getElementById("password");
3 Android.stealCredential(window.location.hostname, uname.value, pword.value);

```

(d) Injected JavaScript for Android WebView

Listing 4.1: WebView Credential Exfiltration Scripts for iOS and Android

aWallet, MyPasswords, PasswordSafe) default to lengths less than 16 characters, which can result in trivially guessable random passwords being generated [48]. Dashlane, Norton, aWallet, PasswordSafe are especially bad, defaulting to only 8 character passwords which are easily broken by modern offline guessing attacks. The only password manager that did not allow users to increase the length to at least 16 characters was Chrome, which only generates 15 character passwords.

In order to help users comply with password policy requirements, five of the password managers studied (1Password, Bitwarden, Enpass, LastPass, Strongbox) allow users to specify the minimum number of characters to include from each character set. 1Password and Bitwarden allow control over only digits and symbols, defaulting to 1 digit and 0 symbols. Enpass allows control over digits, symbols, and uppercase letters, defaulting to 1 of each. LastPass only allows control over digits and defaults to 0. Strongbox does not allow fine grained control, but has a “Pick characters from every group” option that enforces a minimum of 1 character from each character set.

Several additional options were included in some password managers to enhance the usability and memorability of generated passwords. Six (1Password, Bitwarden, Enpass, RoboForm, Strongbox, aWallet, PasswordSafe) allowed users to avoid characters which are difficult to distinguish from each other. Three (Enpass, SafeInCloud, Strongbox) allow user to avoid characters which are difficult to pronounce. Five (1Password, Enpass, iCloud Keychain, SafeInCloud, Strongbox) allow users to generate passphrases composed of dictionary words with symbols as separators, such as “moot-tree-village”. Both Enpass and Lastpass store a history of generated passwords, which increases usability in cases where the password manager generates a password but that password is not properly stored later.

Most of the password managers that we evaluated saved the last used settings as the new default settings, with no way to return to the original default configuration without reinstalling the app. While this behavior could enhance usability, it can also lead to password generation settings being left at those of the weakest password generated, as noted by Oesch et al. [48]. For example, if a user generates an eight character password to satisfy a particular password policy and they forget to change their settings, the next password they generate will also only be eight characters. Only Avast and Keeper returned to default settings each time

the user opened the password generator, but Avast’s default length is less than 16 characters, so it is actually returning to an unsafe default configuration. RoboForm allowed users to restore default settings, but did not do so automatically.

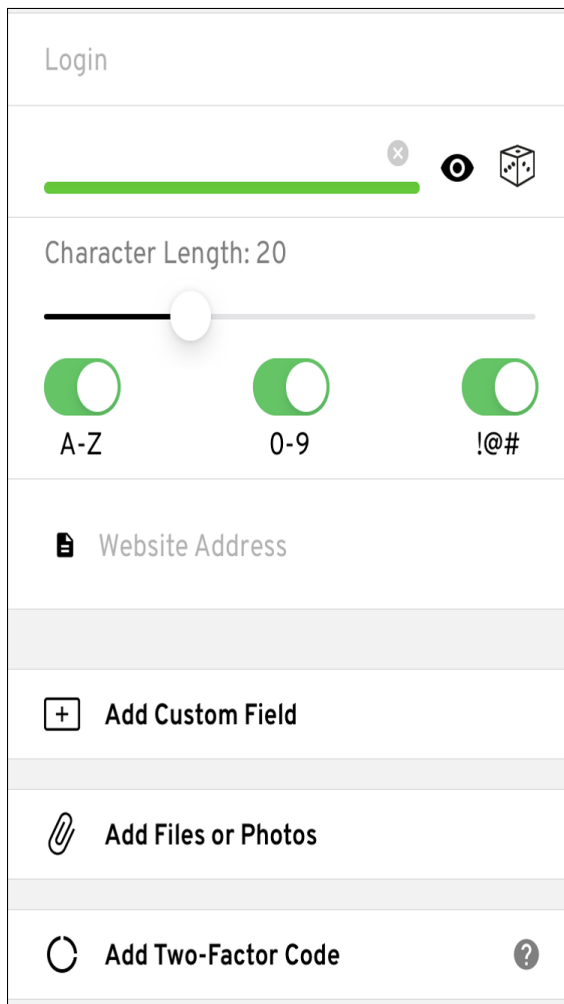
By default aWallet groups character classes—lowercase with lowercase, uppercase with uppercase, symbols with symbols, and numbers with numbers (e.g. “pww!#&XDCJZ6”)—and is the only password manager with this option. This approach can make it easier to enter these passwords on different devices (e.g., a TV) [33], but requires that passwords be longer to provide the same security as passwords without this grouping [43]. In aWallet’s case this is problematic as the tool already defaults to small passwords (8 characters) and does not let users know they need to increase their password lengths when this option is enabled.

4.6.1 Comparison to Desktop Managers

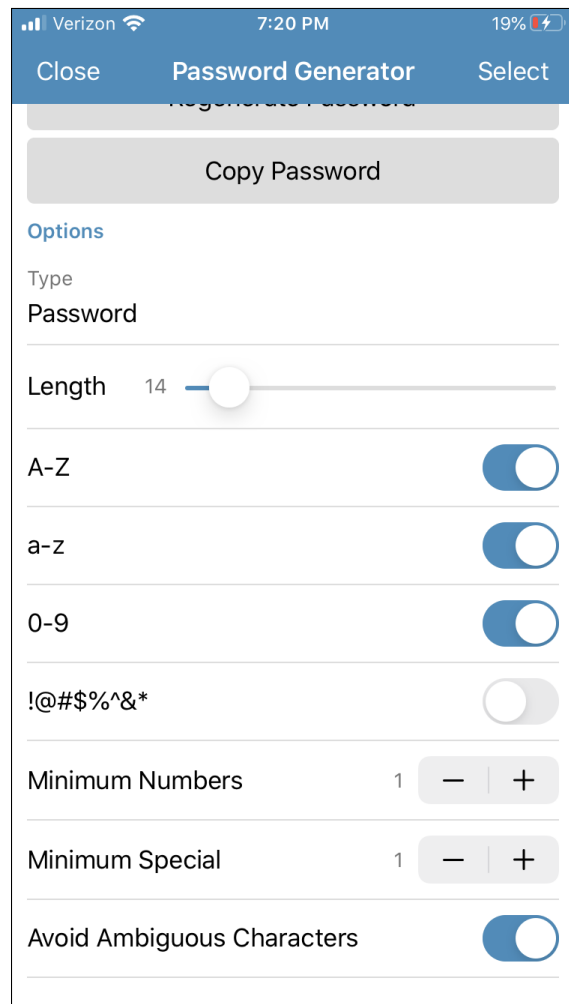
We found a lack of consistency between configuration options available on the desktop [48] and in iOS, even within the same password manager. While this is not necessarily a security issues, it could confuse users when they switch between platforms. In general, we believe that this lack of consistency indicates a similar lack of clearly established best practices for password generation.

4.7 Password Storage

We also conducted an analysis of storage on iOS similar to the analysis we conducted on the desktop, but decided not to pursue the analysis on Android because we did not uncover significant issues. While iOS does prevent apps from reading files (e.g., password vaults) for other apps, there is always the potential that an iOS bug could be leverage to allow a malicious app to read the password vault. As such, it is important that passwords and metadata stored in the vault are properly secured. To evaluate password storage security, we first searched the file system on our jailbroken device to find the password vault file used by each password manager. We then used a combination of approaches (reverse engineering applications, inspecting system calls, looking at source code, communicating with the developer, and reviewing documentation) to determine the encryption algorithm used to protect this file,



(a) Basic (Keeper)



(b) Full (Bitwarden)

Figure 4.6: Examples of password generation configuration.

Table 4.6: Password Generation Features

System		Default composition	Default length	Supported lengths	Set character minimums	Avoid difficult characters	Make Pronouncable	Generate passphrase	Preserve safe settings	Generation history
Autofill	1Password	ld	24	4–64	✓	✓				
	Avast Passwords	ls	9	4–30					✓	
	Bitwarden	ld	14	5–128	✓	✓				
	Dashlane	ld	8	4–40						
	Enpass	ls	50	4–100	✓	✓	✓	✓		✓
	iCloud Keychain	lsd	20	20				✓	✓	
	Keeper	lsd	20	8–51					✓	
	LastPass	lsd	16	8–64	✓					✓
	Norton	lsd	8	4–64						
	RoboForm	lsd	16	4–511		✓				
	SafeInCloud	lsd	12	8–31			✓			
	StrongBox	lsd	16	6–88	✓	✓	✓			
Clip	aWallet	lsd	8	4–20		✓				
	MyPasswords	ld	10	10						
	PasswordSafe	ld	8	1–50						
B	Chrome	ld	15	15						

✓ Default  Optional

including the key derivation function (KDF) used to convert the user’s master password into an encryption key that protects their vault. Table 4.7 and Table 4.8 summarize our findings.

4.7.1 Password Encryption

There were a variety of file types used to stored credentials: SQLite database with encrypted cells (1Password, Avast, iCloud Keychain, Keeper, MyPasswords), a SQLite database that was in an encrypted file (Bitwarden, Dashlane, SafeInCloud, PasswordSafe), SQLCipher (Lockwise, Firefox), and other custom encrypted files (Enpass, LastPass, Norton, RoboForm, aWallet). Chrome and Edge both store their passwords in the iCloud Keychain, but also maintain a separate unencrypted database with additional metadata.

Other than PasswordSafe which encrypts its password vault using TwoFish-256, all password managers encrypt the password vault using AES-256. Five of the password managers (iCloud Keychain, Lockwise, Chrome, Edge, Firefox) generate a random encryption key and store it using iOS’s secure enclave. While the details of the secure enclave are out of scope for this work, we note that it provides adequate (if not ideal) protection for the encryption keys [32]. The other password managers derive an encryption key by feeding the user’s master password to a key derivation function (KDF).

Two password managers (Dashlane, StrongBox) support Argon2D, a memory-hard KDF that is more resistant to offline attacks than PBKDF2. aWallet uses SHA256 as its KDF, though with only 1,000 rounds, which is less than ideal. The remaining password managers use PBKDF2, though several use 1,000 or fewer rounds(Norton, MyPasswords, PasswordSafe), which, similar to SHA256, is less than ideal.

For the password managers that rely on a master password, most do not require strong master passwords, making them vulnerable to offline attack and in some cases even online attacks for vaults that are synced online. Only six (1Password, Bitwarden, Dashlane, Keeper, Lastpass) met the NIST guidelines for passwords [31], though most just barely met these requirements which may not be sufficient for high-value secrets like a master password. Five (Avast, RoboForm, SafeInCloud, aWallet, MyPasswords) allow four character passwords and three (Enpass, StrongBox, PasswordSafe) allow one character passwords.

4.7.2 Metadata Privacy

Other than Chrome and Edge, which do not protect any metadata, all password managers encrypted the username and url associated with a given password. The protection of other metadata was less consistent. Five (1Password, Avast, Keeper, Chrome, Edge) stored the creation and modification times in plaintext and five (Dashlane, Keeper, RoboForm, Chrome, Edge) stored the email associated with the password manager in plaintext. Enpass does store some user settings unencrypted (e.g., KDF rounds), though they are not private and any attempt to change these settings would either cause Enpass to fail to load or be automatically overwritten the next time Enpass was loaded.

Almost half of the password managers (1Password, Avast, Bitwarden, Dashlane, Enpass, Keeper, Norton, RoboForm) store icons for each of the stored credentials. Dashlane fails to encrypt these icons which could allow an adversary to learn what websites a user visits [48]. Four password managers (Enpass, Keeper, StrongBox, aWallet) offer to store attachments for users, though Enpass and StrongBox leave an unencrypted version of the attachment in the `/tmp` folder.

4.7.3 Comparison to Desktop Managers

Compared to desktop password managers [48], we find that browser-based password managers are more secure due to their use of iOS’s secure enclave to store encryption keys for their password vaults. We also find that managers generally use the same KDF settings on both desktop and iOS. In regards to metadata privacy, we find that results are mixed for both desktop and iOS, with some password managers providing good protections and others providing poor protections.

4.8 Discussion

Our analysis provides a mixed message regarding the effectiveness of mobile autofill frameworks. On the positive side, all frameworks enforce user interaction prior to autofill (P1), significantly

Table 4.7: Overview of Password Vault Encryption

		File type	Algorithm	MP KDF	KDF Rounds	Requires strong MP
System		Password encryption				
AutoFill	1Password	SQLite w/ encrypted cells	AES-256	PBKDF2	100,000	●
	Avast	SQLite w/ encrypted cells	AES-256	PBKDF2	10,240	○
	Bitwarden	Encrypted SQLite file	AES-256	PBKDF2	100,001	●
	Dashlane	Encrypted SQLite file	AES-256	Argon2D	3	●
	Enpass	SQLCipher	AES-256	PBKDF2	100,000	○
	iCloud Keychain	SQLite w/ encrypted values	AES-256	Uses secure enclave		
	Keeper	SQLite w/ encrypted cells	AES-256	PBKDF2	100,000	●
	Lastpass	Custom encrypted file	AES-256	PBKDF2	100,100	●
	Lockwise	SQLCipher	AES-256	Uses secure enclave		
	Norton	Custom encrypted file	AES-256	PBKDF2	1,000	●
	RoboForm	Custom encrypted file	AES-256	PBKDF2	4,000	○
	SafeInCloud	Encrypted SQLite file	AES-256	PBKDF2	10,000	○
	StrongBox	KeePass or PasswordSafe file	AES-256	Argon2D	2	○
Clipboard	aWallet	Custom encrypted file	AES-256	SHA256	1,000	○
	My Passwords	SQLite w/ encrypted cells	AES-256	PBKDF2	1,000	○
	PasswordSafe	Encrypted SQLite file	TwoFish-256	PBKDF2	256	○
Browser	Chrome	iCloud Keychain		Uses secure enclave		
	Edge	iCloud Keychain		Uses secure enclave		
	Firefox	SQLCipher		Uses secure enclave		

● Secure behavior ● Partially secure behavior ○ Insecure behavior - Not stored

MP = Master Password KDF = Key Derivation Function

Table 4.8: Overview of Password Vault Metadata Behavior

		Username URL Creation time Modification time User's email User's settings Icons Attachments							
System		Metadata encryption							
AutoFill	1Password	●	●	○	○	●	●	●	-
	Avast	●	●	○	○	●	●	●	-
	Bitwarden	●	●	●	●	●	●	●	-
	Dashlane	●	●	●	●	○	●	○	-
	Enpass	●	●	●	●	●	●	●	◐
	iCloud Keychain	●	●	-	-	-	-	-	-
	Keeper	●	●	○	○	○	●	●	●
	Lastpass	●	●	●	●	●	●	-	-
	Lockwise	●	●	●	●	●	●	-	-
	Norton	●	●	●	●	●	●	●	-
	RoboForm	●	●	●	●	○	●	●	-
	SafeInCloud	●	●	●	●	●	●	●	-
	StrongBox	●	●	●	●	●	●	●	◐
Clipboard	aWallet	●	●	●	●	●	●	-	●
	My Passwords	●	●	●	●	●	●	-	-
	PasswordSafe	●	●	●	●	●	●	-	-
Browser	Chrome	○	○	○	○	○	-	-	-
	Edge	○	○	○	○	○	-	-	-
	Firefox	●	●	●	●	●	●	-	-

● Secure behavior

◐ Partially secure behavior

○ Insecure behavior

- Not stored

improving upon the situation on the Desktop. Additionally, iOS password autofill fully secures the autofill process for native UI elements in apps.

On the other hand, within mobile browsers, all frameworks failed to correctly check credential mapping (P2) and none adequately protected filled credentials (P3), in many cases being less secure than even the worst managers on desktop. Moreover, the frameworks impeded the ability of managers to provide these properties themselves, leading the mobile managers to be less secure than their desktop counterparts. These same issues cropped up for autofill within WebView controls in apps, with other issues leading to our identification of two phishing attacks enabled by the mobile autofill frameworks. Critically, for both attacks, the password manager acts as a confused deputy, displaying the autofill dialog and suggesting that the user fills the credential being targeted by the attack, a dialog which in all other contexts indicates to the user that their credentials are not being phished (see §4.1.2).

Based on these findings, it is clear that current mobile autofill frameworks are not achieving their potential. Still, based on their ability to enforce correct behavior, we do not believe they should be abandoned, but rather fixed to properly secure the autofill process. To this end, we conclude the paper with (a) recommendations for addressing the WebView phishing attacks we identified, (b) providing guidelines for secure autofill framework implementations, (c) identifying framework smells that indicate problematic framework designs, and (d) discussing areas requiring additional research.

4.8.1 Addressing WebView Phishing Attacks

The simplest approach to addressing the WebView phishing attacks would be adopting the proposal from Stock and Johns [60] to only fill credentials into Web requests, not the actual webpage. To do this, the framework would fill fake credentials, then replace those fake values right before they are sent over the wire, effectively preventing JavaScript, and by extension apps, from accessing the filled credentials.

On desktop environments, it is not currently possible to implement this proposal as browsers do not let extensions modify Web request contents and the manager vendors have no control over the browser's internals. In contrast, on mobile the same vendor maintains the autofill framework, the mobile browser, and the WebView control. This integration allows

the vendor to implement Stock and John’s proposal, and we strongly suggest they do so. Note, that implementing this feature would fully enforce P3 for both WebView controls and the mobile browser.

An alternative approach to addressing the WebView phishing attacks would be to modify the WebView so that credentials will only be autofilled if the app has not injected JavaScript. After filling the credentials, the WebView could also have a flag set that prevents the app from injecting JavaScript until a new page is loaded (and by extension the credentials unloaded). Such a prevention mechanism could be loosened for WebViews displayed for domains for which there is an app-to-domain association already existing.

4.8.2 Recommendations for Secure Autofill Frameworks

1. **Require user interaction.** User interaction should be required. While far from a perfect defense (phishing attacks are still possible), it does prevent silent credential harvesting and at least gives users a chance to detect a phishing attack.
2. **Authenticate domains.** For both browsers and WebView controls, the identity of the domain should be cryptographically verified using TLS. This will help prevent network injection attacks from being able to access filled credentials.
3. **Provide a cryptographically verified bidirectional app-to-domain mapping.** Frameworks should require that apps identify the domains they are associated with, and this mapping should be included in the code signing process. Domains should also be required to identify the apps that their credentials can be filled into, and this mapping should use the code signing key’s fingerprint. Only when both these mappings are in agreement should a given domain’s credentials be autofilled into an app.

iOS Password AutoFill already provides this property, and Android could provide this property by expanding their existing DAL-based app-to-domain pairing scheme, enforcing it at the framework level. While currently adoption of DAL is limited—we analyzed 4,081 of the most popular paid apps and 9,345 of the most popular free apps on the Play Store and found that only 10% of paid apps ($n = 402$) and 20% of free apps

($n = 1879$) were whitelisted by a DAL file—we believe that adoption would rapidly increase if Google required such links for apps to be published or updated in the Google Play Store.

4. **Thoroughly evaluate webpages.** While the proposal by Stock and Johns [60] full achieves P3, until it is implemented frameworks should still carefully scan webpages before autofilling credentials to help prevent accidental leakage of credentials. For example, by checking the `action` and `method` fields on the form. Additionally, cross-domain autofill should be disabled.
5. **Thoroughly evaluate forms to be autofilled.** The framework should check the form to be autofilled, ensuring that the password is sent over HTTPS if it should be, that the form is sending the password to the correct destination, and that only valid password fields are being autofilled. These checks should be handled by the autofill framework to ensure that all password managers receive these protections.
6. **Allow password managers to override autofill decisions.** While autofill frameworks should attempt to fully satisfy P1–P3, our research shows that they often fall short. Based on our discussion with manager vendors, we found that many of them were aware of some of the limitations we identify in this paper, but noted that the frameworks prevented them from addressing the issues, most commonly by failing to provide the manager with sufficient information to enforce more secure behavior. This situation could be partially rectified by providing the managers with copious information about the autofill environment and process, then allowing the managers to override autofill decisions as they deem necessary. This would be especially helpful in situations where there are usability and security trade-offs for allowing or disallowing autofill, allowing managers to cater to their individual customer bases.

4.8.3 Autofill Framework Smells

Like code smells, framework smells are a sign that something is wrong with the underlying autofill framework design. These framework smells can be detected by looking at how password managers choose to implement autofill inside a given framework. In our analysis, we identified two framework smells that are strong indicators of poor design.

Warnings. When password managers feel the need to issue autofill related warnings, it may be a sign that the underlying autofill framework is flawed. While the absence of warnings does not imply the framework is well implemented (we encountered almost no warnings on iOS, yet there were still serious problems), their presence may indicate that password manager vendors recognize that certain behaviors are insecure, but they feel they have no other option. For example, Figure 4.7 shows warnings issued by password managers on Android when they could not establish a secure mapping. In this case, the warnings are indicative of the framework failing to satisfy our recommendation that frameworks enforce a secure app-to-domain mapping.

Heuristics. Like warnings, when password managers use heuristics to make security decisions, it may be an indicator that the autofill framework is flawed. For example, our results found that many password managers on Android employ weak heuristics to map apps to credentials, leading to serious security vulnerabilities. Similarly, we see various heuristics used to determine whether autofill is safe on a given form (though far fewer than on desktop where it is easier to implement these heuristics). In both cases, the heuristics only exist because features are missing from the underlying autofill framework.

4.8.4 Future Research

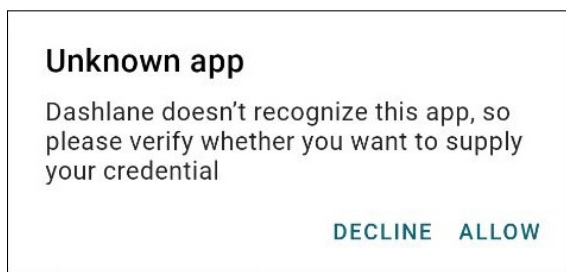
In addition to addressing the issues identified in this paper, we identify three areas of future research.

First, in this paper, we make the assumption that users are likely to fall for the phishing attacks described in §4.5. Based on how easily these phishing attacks can be obscured, the fact that the autofill dialog is supposed to indicate that phishing isn't occurring, and the copious research establishing the ease of phishing users generally [19, 24, 40, 64], we believe

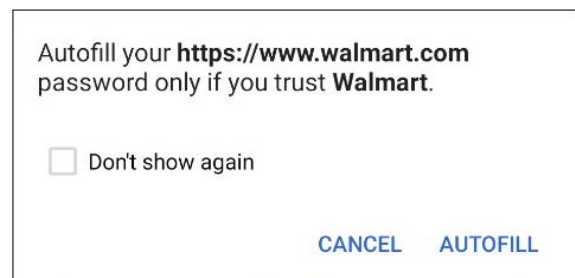
our assumption is sound. Still, future user studies could examine this attack, empirically confirming its feasibility. Moreover, such research may also identify ways in which the autofill dialog could be improved to protect users from phishing attacks.

Second, we believe that research needs to be conducted to design a mechanism that allows a domain to indicate which webpages should be allowed to receive autofilled credentials. This would prevent vulnerable webpages on the domain other than these login pages from being able to steal users autofilled credentials, especially if password managers prevent autofill within same-origin iframes. We believe this feature could be implemented similarly to how mappings work for iOS Password AutoFill or DAL files, having a single file on the website that lists acceptable URLs. Still, research is needed to identify the feasibility and effectiveness of this proposal, along with the best way to implement it.

Third, research is needed on creating an autofill framework for desktop environments. While browsers do provide a platform to deploy password manager extensions, they do not actually provide any password management-centric functionality—i.e., they do not assist with the detection of login forms nor facilitate autofilling credentials. This lack of framework support causes a mixed level of security for password manager extensions. Moreover, there is no OS-level autofill framework, making it nearly impossible for passwords managers to provide universal autofill for desktop applications.



(a) Dashlane



(b) Smart Lock

Figure 4.7: Warnings displayed by Dashlane and Smart Lock when a secure mapping could not be established

Chapter 5

”It basically started using me” - An Observational Study of Password Manager Usage

After analyzing the security of password managers on desktop and mobile platforms, we wanted to understand how and why users use managers, so we conducted an observational interview of password manager users on their own devices. Participants shared their mobile and desktop screens via Zoom and demonstrated how they would normally use their manager as part of a semi-structured interview about their manager usage. We believed that observation was an important component to accurately assess user behaviors and hoped that this methodology would yield new insights. That 6 of our 32 participants reported using a third-party password manager when in fact they used Chrome’s built-in manager and that numerous participants who reported using a single manager actually used multiple managers in practice, sometimes in very unexpected ways, suggests that we were right about the importance of observation. Also, because we observed usage on both participants’ laptop/desktop and mobile device, we were able to identify cross-device issues that led directly to password reuse or significantly impacted usability. For example, some participants reused passwords or created weaker passwords because they knew they would have to enter those passwords on multiple devices, or on devices where their manager would not be present.

The interview contained task-based components where we observed users perform common tasks such as account creation, login, and update, as well as questions related to manager setup and configuration, adoption and promotion, password creation and sharing, and how users identify compromised passwords. Because the interview was semi-structured, we asked additional questions about interesting behaviors as they arose. Participants were recruited from Amazon’s Mechanical Turk by first issuing a screening survey and then inviting selected candidates to participate in the interview itself. Of the 1020 people who completed the screening survey, we invited 456 to participate in the interview, 28 of whom both signed up for and attended the interview. Because our survey design did not change significantly, we also included data from the 4 individuals who took part in the pilot study in our final analysis, for a total of 32 participants.

We followed a four-stage grounded theory approach during our analysis (open coding, axial coding, selective coding, and theory generation). In the initial phase, three researchers reviewed each transcript phrase-by-phrase and assigned codes, resulting in 572 unique codes. In the axial coding phase, we used the constant comparative method to group codes into concepts and then grouped concepts into related categories during the selective coding phase. As we mapped out the concepts identified during coding 5 core theories emerged: multiple manager usage, cross-device entry and reuse, limited usage on mobile, health check desired but overwhelming, and patterns in adoption and promotion.

In our results section we discuss each of these theories in detail, as well as providing the related concept map. We also discuss other relevant results outside of these core theories, such as mental models of password reuse, why the password sharing feature of managers is rarely used, and the impact of COVID-19 on manager usage, among others. Our key contributions include:

- The first observational study of password manager users on their own devices. Also the first study of password manager usage focused on third-party password managers such as LastPass, 1Password, and Dashlane. Prior studies in this space included non-users and users of manager built-in to the browser, with a comparatively small sample size of third-party manager users.

- The persistent availability of browser managers, along with concerns related to losing passwords and cross-device syncing issues, lead users to adopt multiple managers, usually a browser manager in addition their external manager, though sometimes to different managers for desktop and mobile. We are the first to report on this phenomenon of multiple manager usage, which is surprisingly common.
- Users need to enter credentials on devices without a password manager. This leads them to eschew generating passwords, as those are a difficult to enter on such devices. Additionally, fears of not having the password manager available when they need to enter credentials drives users to prefer self-created, but memorable passwords over generated passwords. These results explain the password reuse identified by Lyastani et al. [41], who discovered that reuse still occurs among manager users and especially among those who do not use their manager for the entire password lifecycle (creation, storage, autofill).
- We found that users of built-in browser managers often adopted the manager without any forethought. "It basically started using" them, which explains why users of built-in managers tend to be more focused on convenience than security [72]. They did not look for a password manager to help them manage passwords or based on an advertisement / recommendation that explained why you should use a manager. They just saw the manager pop-up and offer to generate, save, and fill their passwords, so they clicked the pop-up and started to use it. In contrast, users of third-party managers often started using them at work or based on a recommendation, both places that are more likely to explain how a manager is intended to help you avoid reuse and demonstrate proper usage.
- Usage of password managers on mobile devices is very limited. The reasons for this are far-ranging: (1) inconsistent autofill and autosave functionality, (2) issues syncing between desktop and phone password managers, (3) apps and websites that stay logged in near-permanently on mobile, and (4) preference for SSO. Even if users do use a mobile password manager, they often do not use password manager features such as autofill and autosave, instead opting to copy and paste or manually enter passwords.

5.1 Methodology

Prior user studies of password managers rely on user reported behaviors collected via interviews, surveys, and post-task questionnaires, and tend to focus on either mobile or desktop/laptop usage. In contrast, we conducted the first ever cross-device observational interview, in which we observed password manager users complete a set of tasks on their own devices via Zoom as part of a semi-structured interview. This IRB approved study took place from January 25th to March 4th 2021, with 1020 individuals completing the screening survey and 28 participating in the interview. Because our survey design did not change significantly, we also included data from the 4 individuals who took part in an IRB approved pilot study in our final analysis, for a total of 32 participants. Below we describe our recruitment process, interview design, and analysis procedure, as well as participant demographics and methodological limitations.

5.1.1 Recruitment

We recruited adult participants (aged 18 and over) using Amazon Mechanical Turk in two phases. First, we sent out a screening survey (see Section E.1) to collect demographic data, including which password manager the respondent used, and answers to questions regarding password generation from the security behavior intentions scale (SeBIS) [20]. We also included select questions from Masur’s online privacy concerns scale [42] that addressed both vertical and horizontal privacy concerns. We initially tried to recruit Master Workers, those who have consistently performed well across a variety of tasks, to ensure high quality responses, but after receiving 101 responses from January 25th to January 28th 2021, only 18 of whom indicated they would be willing to participate in a follow-up interview, we decided to include non-Master Workers as well. By January 30th we had collected a total of 1026 responses to the screening survey. We had to drop 6 responses because even though they completed the survey, they failed to provide the survey completion code, so we had no way to link their response to their Mechanical Turk ID, resulting in 1020 valid responses. Participants received 1\$ in compensation for completing the screening survey.

Next, we invited respondents to participate in the interview itself by completing a second brief survey (see Section E.2) that included signing up for a Zoom interview, as well as questions related to general security habits, device security, and software updates from the SeBIS [20]. Initially we binned respondents into 6 buckets based on gender and age with the goal of inviting an equal number of people from each bucket to participate in the interview. But we soon altered our approach when only 3 of the first 24 people we invited actually signed up for the interview and of those 3, 1 did not attend the scheduled appointment. Ultimately, we ended up inviting 456 people to participate in the interview, with 28 actually participating. We stopped at 28 participants because we believed we had reached saturation. We conducted these interviews from February 16th to March 4th, 2021. On average interviews took 35-40 minutes and participants received 26\$ in compensation. Consent forms for both the screening survey and the interview can be found in the Appendix (Sections E.5 and E.4, respectively).

The 4 pilot study participants were recruited via convenience sampling and were all researchers and/or engineers who worked with one of the authors. An email was sent out inviting coworkers who used password managers to participate in the pilot study, and those who responded in the affirmative were included. Only minor modifications were made to the interview design as a result of the pilot study, mostly related to ensuring participants were able to share their screen on all relevant devices and utilize the provided temporary account for their password manager prior to beginning the recorded portion of the interview. The pilot studies took place from January 6th-13th, 2020. Throughout this chapter, we refer to pilot study participants with a 'P' prefix (P1-P4) and MTurk participants with an 'R' prefix (R1-R28).

5.1.2 Interview Design

We designed the interview so that we observed participants using their password managers on their own devices via Zoom. Whenever possible, we asked participants to demonstrate specific behaviors during or prior to questions regarding those behaviors. The interview itself was semi-structured and while the interviewer followed a guide (see Section E.3) walking them through what questions to ask and how to guide the participant, they were encouraged

to engage in fruitful dialogue and explore any interesting use cases that arose. For continuity, a single researcher conducted all 32 interviews.

Participants were given an overview of the purpose of the research and the interview structure, as well as a reminder that the interview would be recorded. To encourage participants to accurately represent their behaviors, we reminded them that any problems they were having reflected issues with password managers and not with them personally. We also encouraged participants to interject whenever something they thought would be helpful occurred to them.

Prior to starting the recorded portion of the interview, we made sure participants were able to share their screen from both their desktop and mobile device (if they used mobile). We also had participants log out of their password manager on all relevant devices to protect their privacy before we started recording. Once we ensured that the participant was comfortable with sharing their screen via Zoom from all relevant devices, we began the recorded portion of the interview.

First, we asked questions related to general usage, such as how participants protected their password manager accounts, such as how they created their master password and whether or not they use 2FA. We also asked how they chose this particular password manager, whether they ever recommend it to others, and if others adopted it based on their recommendation. Then we asked if they ever saved passwords anywhere other than their primary password manager, such as in the browser or on a notepad.

Second, we had participants login to their password manager with a temporary account that we provided and show us if there were any settings they would change when first configuring their manager on a new device. We then had them create an account on both reddit and ebay, walking us through how they would normally create an account. During account creation, we asked participants if they had issues when creating or saving accounts, and about their password creation strategies.

Third, we had participants log back in to those same two accounts, asking questions about how they utilize autofill. Once participants had logged in, we had them update their password for both accounts. We then asked them questions about their password updating habits.

Fourth, we asked participants about their password creation and storage habits for different types of websites, their use of the autolock feature of their manager, and whether or not they filled passwords into apps on their desktop. These questions often led to fruitful discussions around important versus nonimportant accounts, device privacy, and edge cases for autofill.

Fifth, we asked participants how they normally go about sharing passwords and, if they were not already aware of it, demonstrated the password sharing feature of their password manager. We also asked how participants would normally check if their passwords had been compromised in a data breach and demonstrated the health check feature of their manager if they were unaware. Both of these questions led to extended discussions around password sharing and auditing behaviors. We then asked participants if there were any additional features they used that we had not yet mentioned (e.g. secure notes)

At this point we had participants stop sharing their desktop screen and pivoted to their mobile device. Regardless of whether they used their password manager on their mobile device, we began with an open ended question about how password management was different on their mobile device than on their desktop. We then asked them whether they created accounts on their phone, how they logged into websites and apps on their phone, and if they created/generated passwords on their phone.

If they used any password manager, including the one built into their phone's OS, on their mobile device, we had them share their mobile screen and demonstrate those behaviors they regularly practiced. Depending on their answers to prior questions, we had them login to reddit using the account we just created to help us understand how they sync between devices and how they utilize autofill on mobile. We also had them create, log out of, and log back into an account in the memrise app if they used a manager for apps.

Finally, we had them stop sharing their mobile screen, if necessary, and wrapped up with a few closing questions. The first question was open ended and provided participants an opportunity to share anything else they felt we would find helpful. We then asked what feature they would add or thing they would change about their manager, if any, and closed by thanking participants for their time and explaining compensation.

5.1.3 Addressing Potential Risks

Because we did observe participants using their own personal devices, we took several steps to protect their privacy. First, we reminded them at the beginning of the interview that we would be recording them on their own devices and gave them an opportunity to opt-out. Second, we had users log out of their password managers before we started recording, and then had them login to a temporary account that we provided. Third, we had users disable their video feed prior to recording just in case it exposed any private information. And fourth, when we uploaded the interviews for transcription, we only uploaded the audio just in case any private information was exposed in the recordings.

We also note that there was a potential benefit for participants. Many participants said that they learned new things about their manager through this interview. And after the interview was over, we were able to help several participants understand their manager better, which was particularly important for several of the participants who had recently had an account breached.

5.1.4 Analysis

We used an automated online transcription service to transcribe the interviews, followed by a pass from one of our researchers to verify accuracy. Only the audio file was uploaded to the transcription service to avoid the possibility of leaking any sensitive information contained in the video we may have overlooked. The transcribed interviews, along with artifacts of the analysis, are available at [redacted].

We followed a four-stage grounded theory approach during our analysis (open coding, axial coding, selective coding, and theory generation). In the initial phase, three researchers reviewed each transcript phrase-by-phrase and assigned codes. Codes were primarily assigned via open coding (the code summarizes the participant's statement), though some in situ codes were generated (using the participant's own words as the code). We used contextual clues to ensure our codes were accurate, as well as revisiting the video recordings when uncertain to more closely examine participant actions and hear their tone. After completing open coding, we had a total of 572 unique codes.

In the axial coding phase, we used the constant comparative method to group codes into concepts. When two codes were describing the same phenomena, we collapsed them into a single code. Otherwise, we grouped codes underneath concept headings rather than collapsing them so that we could easily revisit what codes made up a concept and move codes back and forth between concepts as we iterated.

In the third stage, we grouped concepts into related categories, drawing and labeling connections between both concepts and categories using concept maps. Five key categories emerged from this analysis: multiple manager usage, cross-device entry and reuse, limited usage on mobile, health check desired but overwhelming, and patterns in adoption and promotion. We discuss each of these categories in detail and provide the concept maps for each in Section 5.2.

Finally, we used the categories and connections between them, the concept maps, and our research notes and observations to generate theories about how people use their password managers. We developed one theory for each of the five categories that we identified, as well as exploring the implications of our other results. Given that these theories are based on observational interviews with 32 individuals, they are not conclusive, but are grounded in the data that we collected.

5.1.5 Participant Demographics

Our population was roughly 2/3 male (22/32 participants) and was predominantly composed of individuals between the ages of 21 and 44, with only 4 participants 45 or above. We had 23 participants who identified as Caucasian or White, 4 as Black or African American, 4 as Asian, and 1 as Hispanic or Latino. A majority of our participants had either a Bachelors (11), Masters (8), or some college (5). One participant held a doctorate, and several had either an Associates (2), High School or GED (2), or a Professional degree (3). Following an approach similar to that of Tan et al. [61], we considered participants technical if they reported that people asked them for computer-related advice and that they knew a programming language. Approximately half of our participants were technical by this measure (17 in total).

5.1.6 Limitations

Because interviewees may desire to appear knowledgeable to the interviewer in the context of a semi-structured interview, it is possible they reported security practices better or different than their normal behaviors. [2] Likewise, it is possible that direct observation altered participant’s behavior [51]. We encouraged participants to behave naturally by observing them on their own devices, verbally nudging them to do what they would normally do, and above average payment, however, we cannot exclude that some participants behaved unusually.

We also targeted MTurk users who live in the US, a group that tends to be younger, better educated, and more privacy conscious than the general population [37]. Future work could replicate this study with different populations, as well as use more quantitative, large-scale approaches to explore specific observed phenomena.

5.2 Core Theories

We used grounded theory to identify five theories in the data from our interviews that explain how people use password managers. In this section we explain each of these five theories, provide the associated concept map that visualizes these theories from our data, and discuss additional important topics that arose in our study. Significantly, several of these theories describe phenomena never before observed in prior research, such as the simultaneous use of multiple managers, and shed new light on known issues, such as password reuse among password manager users.

5.2.1 Adoption of multiple managers

This study was designed with the expectation that users of external managers such as LastPass, 1Password, or Bitwarden would only use that manager. However, by observing participants on their own devices we found that the usage of multiple managers in tandem, generally a browser manager (e.g. Chrome) in addition to an external manager (e.g. LastPass), was very common among our participants. We also observed some participants using a different

manager on their phone versus on their desktop. To our knowledge, we are the first to report on this phenomena. Figure 5.1 shows the concept map for multiple manager usage, highlighting the causes and resultant behaviors.

The most common instance of multiple manager usage was storing passwords in both the browser and external manager. This behavior was motivated by three key factors: (1) Participants scared of losing their passwords used the browser as a failsafe, (2) Participants stored their most commonly used websites in both managers for convenience, (3) Participants used the browser manager without forethought because it popped up and offered to save their credentials. The below quotes demonstrate behaviors (1) and (3).

Browser as failsafe - R9: *“I do have them (passwords) in Chrome as well, only because I have this horrible phobia that I’m going to, like LastPass or something like, oh, no, something’s gonna blow up. Like, I’m going to forget the password and not be able to sign in, something’s gonna happen. And I kind of want to just have a backup. Oh, I just I’m paranoid like that.”*

Storing in browser without forethought - R13: *“I guess when I said I would click on and save it (in browser manager). It’s sort of just out of habit of clicking a popup. I’m kind of always like, just get the pop up out of here.”*

The quote from R13 demonstrates some annoyance with the autosave dialogue popup generated by the browser and points to a common theme among users who utilized multiple managers on the same device. One manager often interfered with the other one because it duplicated functionality. The below quote from R12 demonstrates this type of interference with both the autofill and autosave features, which actually led to this participant’s two managers being out of sync with one another.

Browser manager interferes external manager - R12: *“Yeah, if it autofills, it’s kind of like first come first serve. If Google (Chrome) gets in there and autofills it before I have the chance to tell LastPass. It does cause some problems. So when I’ve updated a password somewhere out there in Google and now Google is out of sync with LastPass.”*

We also found that some participants used different managers on their phone and desktop, as demonstrated by the quote from R17 below, or used multiple managers on their phone. These phenomena are tied to the reality that managers are used in only a limited fashion on mobile devices, which we will discuss in more detail later. However, it is worth noting here that the main reason participants used multiple managers on their phone is that the manager built-in to the OS (Apple or Android) offered to save their passwords and even though they had not originally intended to save it in their phone's manager they did so just to get past the popup (see quote from R12 below).

Different manager on desktop and mobile - R17: *"Yeah. So I have an iPhone and on my phone, I use the apple password manager for everything, passwords, credit card information, debit card information. And then on my computer, my desktop or my laptop, I actually use Chrome."*

Unintentional usage of phone manager - R12: *"I do but not intentionally. Chrome picks them (passwords) up sometimes. And I don't stop it is what it boils down to."*

Using multiple managers led to situations where participants had to manually enter their passwords to ensure they were saved in both managers, or autofill with one manager and then save with the other (when both managers were on the same device). These frustrations with syncing between multiple managers led R17 to share the following:

Multiple manager usage leads to syncing issues - R17: *"I would want to make it so that like, Apple and Chrome could marry one another, like, oh my goodness, you know, like so passwords can transfer easily."*

The most extreme case of multiple manager usage was R7, who used Safari at home, Chrome at work, and aWallet on their phone. They used Safari because it synced passwords between their personal devices, Chrome because they used PCs at work, and aWallet for their most sensitive accounts and accounts that they needed to access when not near their personal computer. Several participants did intentionally avoid using multiple managers, either because they recognized it was unsafe to store their passwords in multiple locations

or because they felt the browser manager was less secure than their external manager. And some participants who felt it was unsafe to store their most sensitive accounts in a manager stored them in plaintext (Word file or in a notebook), which could be considered another form of multiple manager usage.

5.2.2 Cross-device entry and reuse

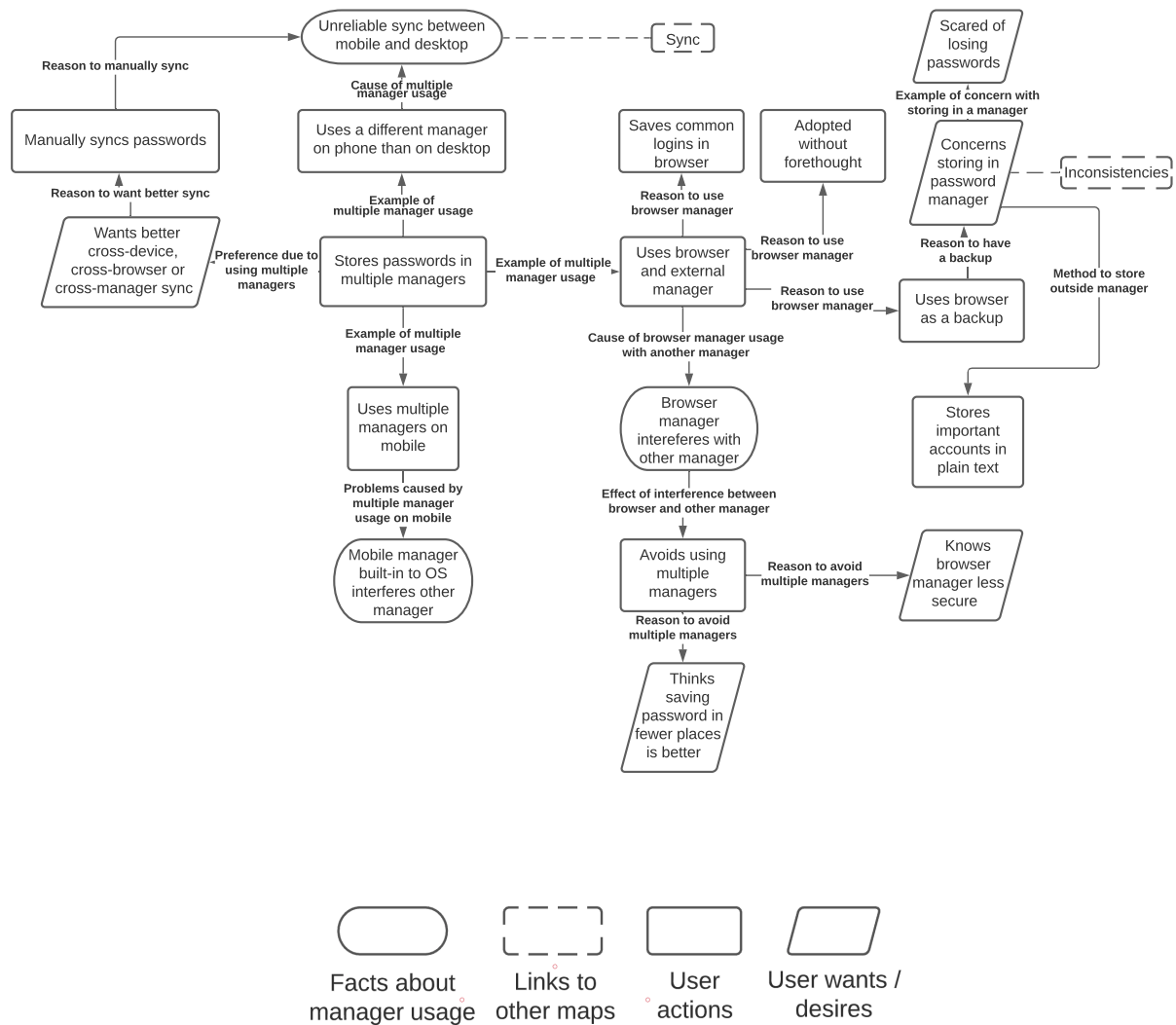
Users need to enter credentials on devices without a password manager. This leads them to eschew generating passwords, as those are difficult to enter on such devices. Additionally, fears of not having the password manager available when they need to enter credentials drives users to prefer self-created, but memorable passwords over generated passwords. The following quotes from R4 and R21 demonstrate each of these concerns, respectively, and Figure 5.2 provides the concept map exploring password reuse.

Generated passwords hard to remember and enter cross-device - Interviewer: *“If I told you that it’s not really secure to create passwords the way that you’re creating them, but it was more secure to use the password generator, do you think that would motivate you to start using the password generator?”*

R4: *“It would not. And the reason for that is that the password generator will give you like a bunch of gibberish letters, there’s no way you could ever remember it. So let’s say if you’re outside and you’re on your mobile and you want to, you know, go to the website that you haven’t been to on on the phone, chances are like good luck remembering that password, you would never log in.”*

Fear of not having manager available prevents use generated passwords - R21: *“Like I’m very, very set in my ways in which passwords I use. So if they have too many, you know, special characters or numbers, I feel like not going to use that. So just in case the Chrome password manager craps out on me, it’ll be easier for me to remember.”*

However, we also found that in some cases the password manager did stop reuse and result in the use of the generator to create strong, unique passwords. Figure 5.3 provides the



Half of participants used multiple managers due to issues syncing between devices, fear of not having their main manager available, and simply because the manager built-in to the OS or browser was persistently available. Only a few participants specifically avoided multiple manager usage, generally due to concerns about saving their passwords in too many places.

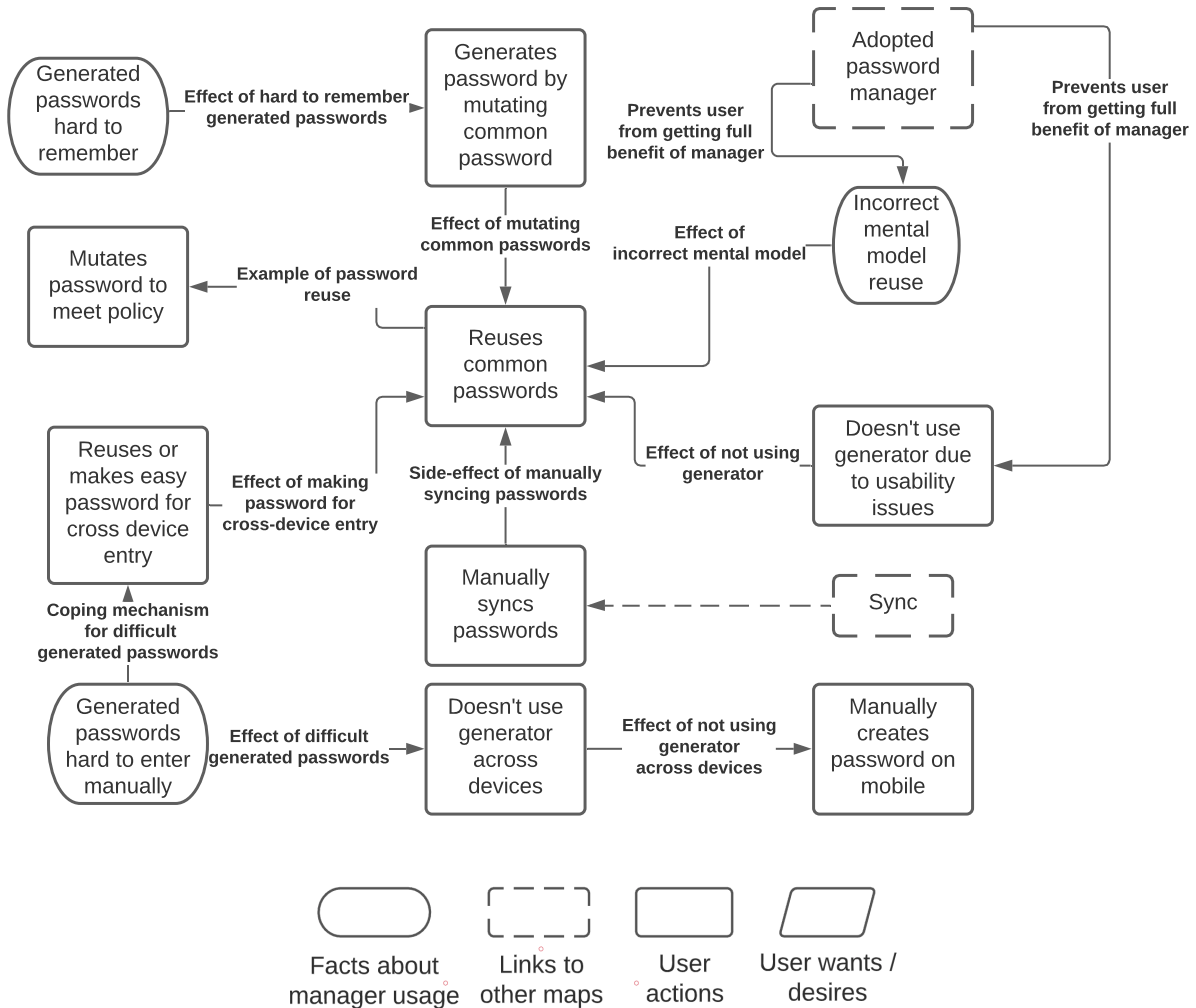
Figure 5.1: Concept map for multiple manager usage.

concept map exploring the cause and effects of stopping reuse. Generally this occurred when the person adopted the password manager specifically to stop reuse (adopted for security rather than convenience) or when the password manager was so usable cross-device that they were not concerned about the availability of their passwords. An example of usability driving the creation of strong passwords is given below, where R10 used all Apple devices and found that the passwords synced between them seamlessly, which gave them no reason to question whether or not it was safe to use the generator.

Usability of password manager leads to using generator to create strong passwords
- R10: *“Years and years ago, um, you know, I tried to use a system or like, I’d use the same base password and kind of modify it, depending on the site... You know, currently using keychain, I just let it generate the password and hit OK... And I really like the password generation feature to which I mean, I know that’s not exclusive to keychain, but since it can do it like right in the browser, wherever I’m working. It’s very beneficial. And then, you know, it just works well between my phone and my iPad and my computer where I don’t have to worry about like syncing them or anything like that. It just kind of natively does the work for me.”*

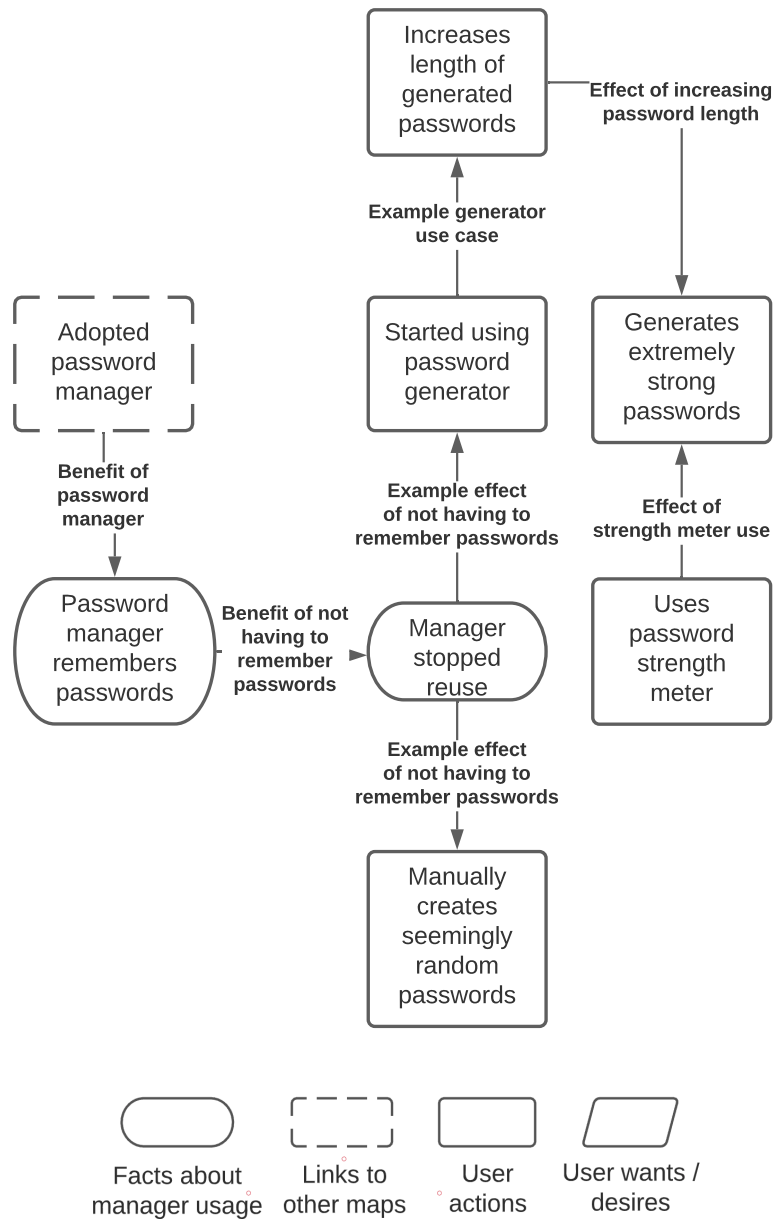
In some cases participants used the generator even though it made it difficult to enter passwords on other devices and different participants had different coping mechanisms. For example, R28 (quote below) created easier to enter passwords without the generator if those passwords would be entered on alternate devices. Another user who used all Apple devices but did not use keychain as their manager utilized AirDrop to sync passwords and a number of participants simply typed them in manually, though most participants who entered manually avoided the generator altogether.

R28: *“I want to say though I do have like one or two that are ones that I made more frequently logging in from a separate device and don’t want to deal with Dashlane, where those have a similar thing of a sentence that I remember with some added digits on top of it.”*



Password reuse was common in spite of the fact that all participants used a manager. While several users simply did not understand why reuse is bad, we found that most reuse was due to usability issues with cross-device entry. Users felt generated passwords are difficult to remember and enter manually on other devices, so they reused common passwords instead.

Figure 5.2: Concept map for password reuse and its causes



Some participants adopted a manager specifically to stop their own reuse of passwords or stopped reusing after adoption, instead using the password generator. Of those participants who used the generator, some increased the length of generated passwords or used the strength meter, resulting in extremely strong passwords.

Figure 5.3: Concept map for when managers stop reuse.

5.2.3 Limited usage on mobile

Among our participants, usage of password managers and their features on mobile devices was very limited. The reasons for this are far-ranging: (1) inconsistent autofill and autosave functionality, (2) issues syncing between desktop and phone password managers, (3) apps and websites that stay logged in near-permanently on mobile, (4) limited usage of mobile device, and (5) preference for SSO. Even if users do use a mobile password manager, they often do not use password manager features such as autofill and autosave, instead entering passwords manually into their mobile manager and then copying and pasting into login fields. Figure 5.4 provides the concept map exploring mobile usage and inconsistencies in manager behavior.

First of all, it is important to recognize that syncing between desktop and mobile is not intuitive for many users. As demonstrated by the quotes below, some participants did not even have a method for saving accounts that they created on their phone, while other participants would manually enter those credentials into their desktop manager at their earliest convenience. For those participants concerned about security, syncing issues actually led some to create easier to remember passwords if they made a new account on their phone while away from home, and then create a stronger password once they got home and could save those credentials in their desktop manager. Several participants, such as R2 quoted below, were not even aware that their manager offered a mobile solution—their mental model of password managers simply did not include the idea of a mobile manager.

Not saving accounts created on mobile - Interviewer: *“How do you remember the password the next time? (after creating an account on mobile)”*

R25: *“I don’t. That’s the problem. Yeah, I don’t like because right now like Facebook, I know my password. So I can log in if I wanted to. But like say I didn’t know the password. I would have to just go back to the app or go onto my computer or reset it or something like I just wouldn’t be able to.”*

Using desktop manager to save mobile credentials - R2: *“And as I said, like, I don’t use LastPass on my phone, yet, I’ll go on my computer to see if like a certain app password is saved on LastPass. And that’s been helpful to log into a*

lot of things that I normally would have had to change the password, because I would never have remembered what symbols were in them.”

Other key underlying factors behind limited usage of mobile managers and their features include limited usage of mobile in general (see R21’s response below), the fact that some people remain logged in near permanently on mobile (see R19’s response below), and a tendency to only use the mobile manager to sync passwords from the desktop and then copy and paste rather than utilizing autofill or autosave (see R1’s response below).

Only visits limited websites on mobile - R21: *“And I started doing it on that to some degree, but there’s only like, probably less than 10 websites where I have my password saved on my phone. Like, my laptop has a lot more.”*

Logs into websites infrequently on mobile - R19: *“I don’t log into sites nearly as often on my phone. When I first set up the phone, I used it to actually authenticate into Reddit and authenticate into other things. But those remain logged in throughout the lifetime of the app as long as you’re not, you know, logging out of it.”*

Only uses mobile manager as reference book - R1: *“But in the sense I only use it as like a reference book. If I had a book with me, with all my passwords in it, I would look one up. That is what I use it for.”*

Another factor preventing the usage of mobile manager features such as autosave and autofill is that they are inconsistent on mobile, so participants are forced to adapt other strategies when these features fail and may abandon these features altogether. While in many cases autofill or autosave simply did not work with a specific app or website, in others autofill did not offer the stored credential for security reasons. iOS checks a special digital asset link file on a website before autofilling an app to ensure that the app is allowed to access those credentials, and it will not offer the credentials otherwise. However, none of the participants understood that this app to website mapping even existed or that there was such a security feature, so they simply perceived this behavior as an annoying inconsistency. As expressed by R11 below, these inconsistent behaviors caused frustration for many users, and often resulted

in users copying and pasting credentials and manually creating accounts rather than trying to rely on inconsistent manager features.

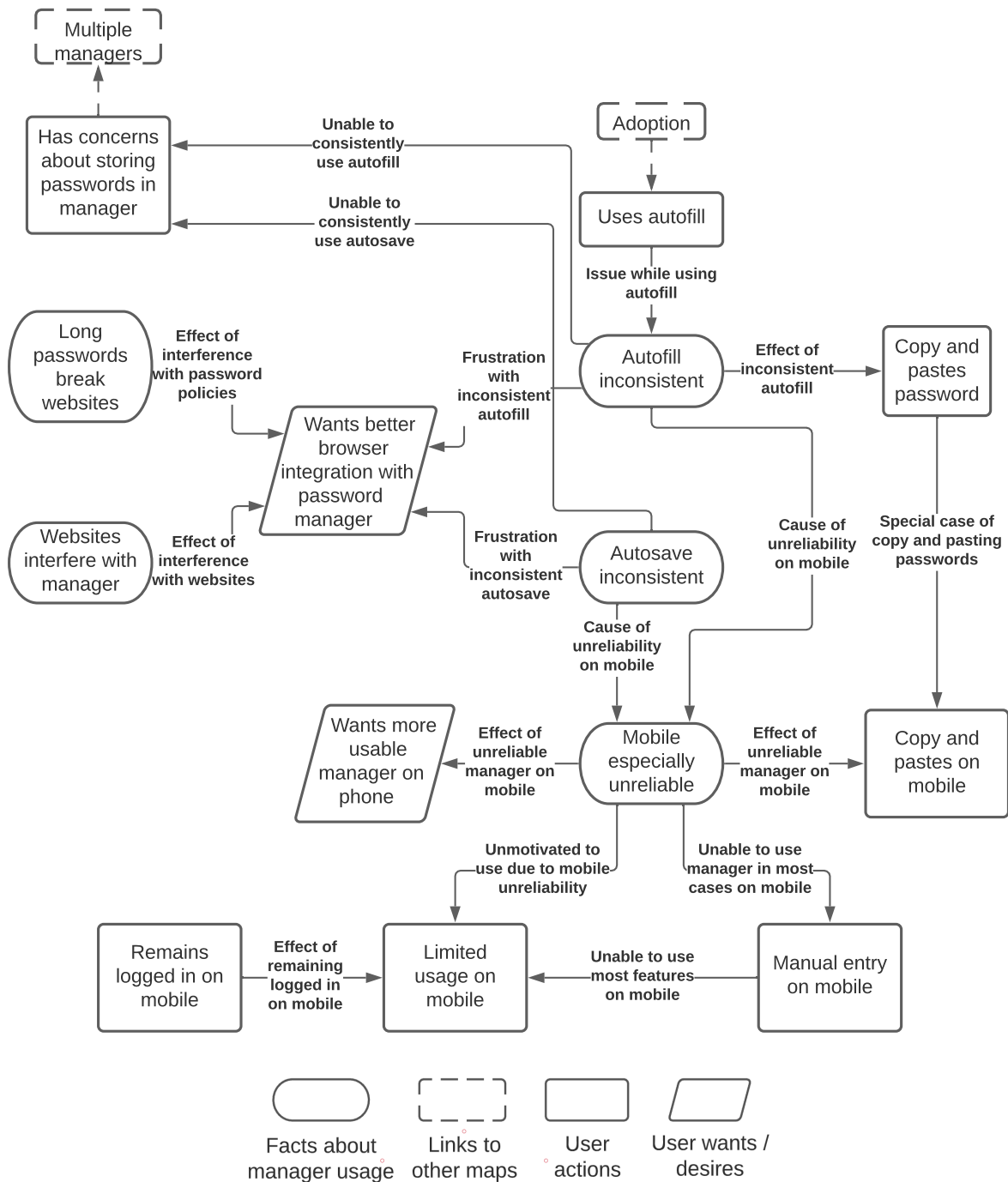
R11: *“My biggest annoyance with the Google and the Android OS password system is its inconsistency. It’s just it sometimes works, and sometimes it doesn’t. Sometimes it does what you expect it to do and other times it doesn’t.”*

5.2.4 Breach notifications desired, but health check overwhelming

Users are interested in learning when their credentials have been compromised and are willing to change them for accounts they consider important. For this reason, participants appreciate breach notifications informing them when important accounts are compromised. However, users often find the health check warnings produced by password managers to be overwhelming because they show dozens of weak/reused passwords and breached accounts, leading them to dismiss those warnings as opposed to figuring out which accounts they do need to update. (i.e., targeted better than fire hose). The below quote from R12 demonstrates this tension between feeling overwhelmed and wanting to keep accounts safe, and Figure 5.5 explores users approaches to auditing their accounts and health check.

Health check overwhelming and therefore ignored - R12: *“It’s a very cool feature. However, it’s, you know, the duplicate password thing, like, there’s a bunch of passwords I’ve never changed, because they don’t have any financial implications for me. And so it just comes up every time. And so it’s kind of become background noise for me, that when I click in, it pops up and says, “Hey, you have a duplicate password here”. And then I go, oh, yeah, I gotta change that. And that’s been about, I don’t know, four or five years? I don’t know. No, I don’t remember how long anyways, it’s a lot, you know, it’s just white noise at this point, I just kind of dismiss it immediately. And there’s an option to turn it off permanently. But I don’t want that because someday I’m actually gonna take care of it.”*

Participants generally fell into three categories regarding their approach to updating their accounts: (1) Those who sensed they should update their accounts more often but only did



Users generally did not mind inconsistencies in autofill and autosave on the desktop, though they did want their manager to integrate better with their browser for a variety of reasons. However, mobile managers were especially unreliable, to the point that users often resorted to manual password entry and copy and paste. Given that many users remained logged in on mobile anyways, some users simply avoided the mobile manager altogether.

Figure 5.4: Concept map for limited manager usage on mobile devices.

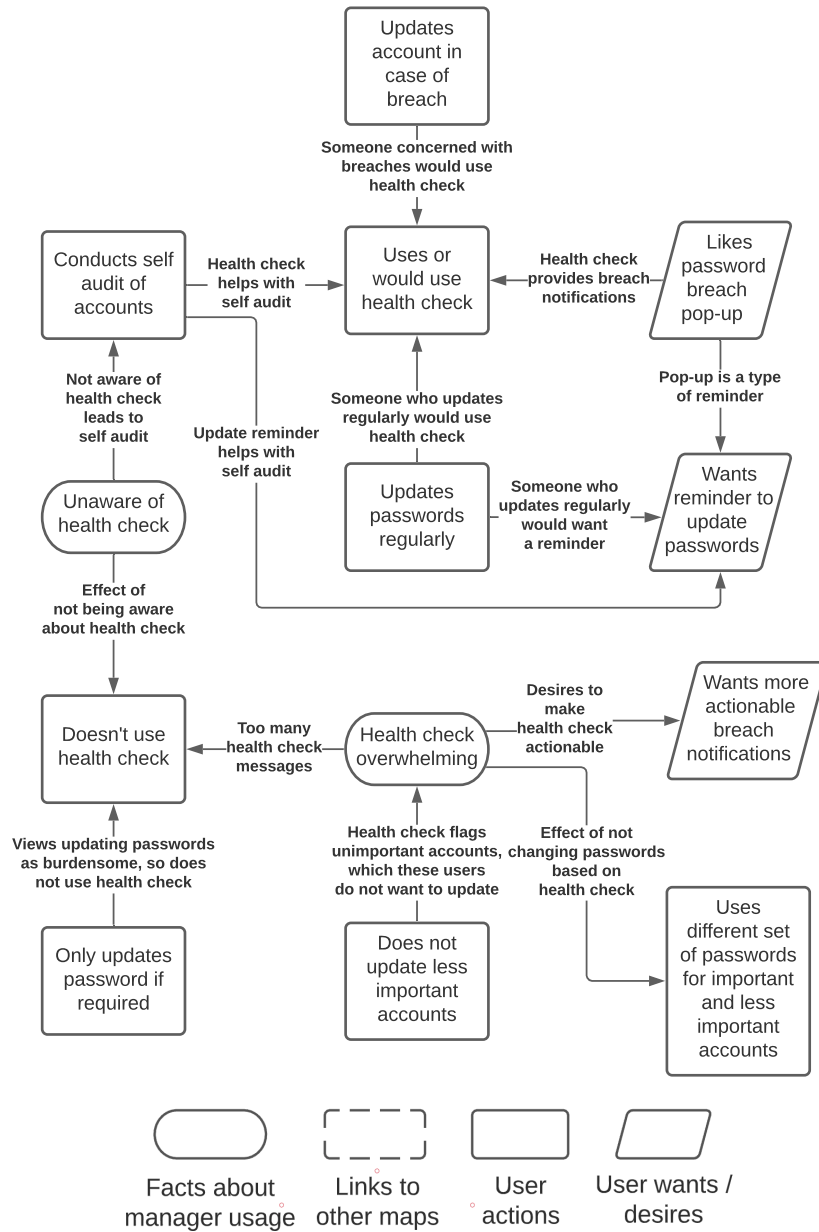
so if required (see quote from R12 below), (2) Those who periodically audited their own accounts, both by updating passwords (R26 below) and/or removing unused accounts from their manager (R11 below), and (3) Those who found updating passwords to be annoying and did so only if required. Those participants who fell into category (1) or (2) liked being notified when their accounts were breached, but many still found health check to be too noisy / overwhelming to use it frequently. In contrast to health check, which required participants to navigate to another page in the UI, many participants liked Chrome's password breach notification popup, though they still found this overwhelming. Overall, these results suggest more targeted popup breach notifications that could be disabled would be welcome and satisfy users in all three categories.

Interviewer: *"And how often would you say you, you update passwords?"*

Only updates passwords if required - R12: *"It's not I'm not proud of it, but about never. Yeah. things require me to basically, which a lot of important places do require me to banks and credit cards and things like that. But even they may be only required once a year, you know? Where probably should be frequent than that."*

Self-auditing - updates important accounts periodically - R26: *"I would say maybe once or twice a year. I should probably do it more often, but... I would say typically the high priority ones, and then if there's something that just prompts me to change it for the other ones, I might do it as well, you know, you just read that, you know, there was a compromise of you know, 20,000 accounts. For example, I might just decided to change it as well, even if it's not too hypercritical."*

Self-auditing - removes unused websites - R11: *"But there's a lot of those websites aren't around anymore. So I do I need to remove them. So, usually I've gone through, I have over 200 (accounts saved) and I've knocked it down to 97. So but most of the websites are either gone, or I don't use them anymore, so I removed the account entirely."*



While some users only cared about convenience and did not update passwords even if their accounts were breached, most users would update passwords for breached accounts and found breach notifications helpful. However, health check results were often overwhelming when first checked because users had so many weak/reused passwords. The result was that most users simply avoided health check. Users who self-audited accounts were the most likely to use or want to use health check in its current form.

Figure 5.5: Concept map for account auditing and health check.

5.2.5 Adoption and promotion

Users generally adopt a password manager for three reasons: (1) requirements to use a password manager at work (see quote from R16 below), (2) it makes their life easier, and/or (3) it allows them to increase the security of their online credentials. Most users like their password managers, but only about half of participants were active promoters. For those that are active promoters, success is limited, with most of that success coming from family members or very close friends. When selecting a manager, users rely on online or personal recommendations (see quote from R26 below). Figure 5.6 provides the concept map for adoption and Figure 5.7 provides the concept map for promotion.

Adoption due to work - R16: *“Actually, I didn’t pick it. My job started forcing it. And then I just got used to it. I might as well just put it on my phone.”*

Adoption based on recommendation - R26: *“That was actually pretty much the first one that kind of popped up. You know, when I was looking for password manager (online). I did speak with friends and there is one friend that was using it and had just a good personal experience with it.”*

Interestingly, several active promoters of password managers felt that they were unsuccessful because other people did not have an accurate mental model of how a manager works. For example, R23 below struggled to help people understand that if they used a password manager they would only need to remember a single password (not many), and struggled to convey why reuse is bad even if the password you reuse is “hard to guess”.

Interviewer: *“Why do you think no one adopts a manager when you recommend it?”*

R23: *“No one understands utility, or they just can’t wrap their mind around how they would use a password manager because they’re always trying to think well I have to remember all these different password No, you don’t remember one password. Yeah, that’s why I think I have no converts.”*

P1: *“A lot of times, it’s something to the effect of no one would guess my password. So why worry about it? And then you explain, algorithms don’t care about whether it’s hard to guess.”*

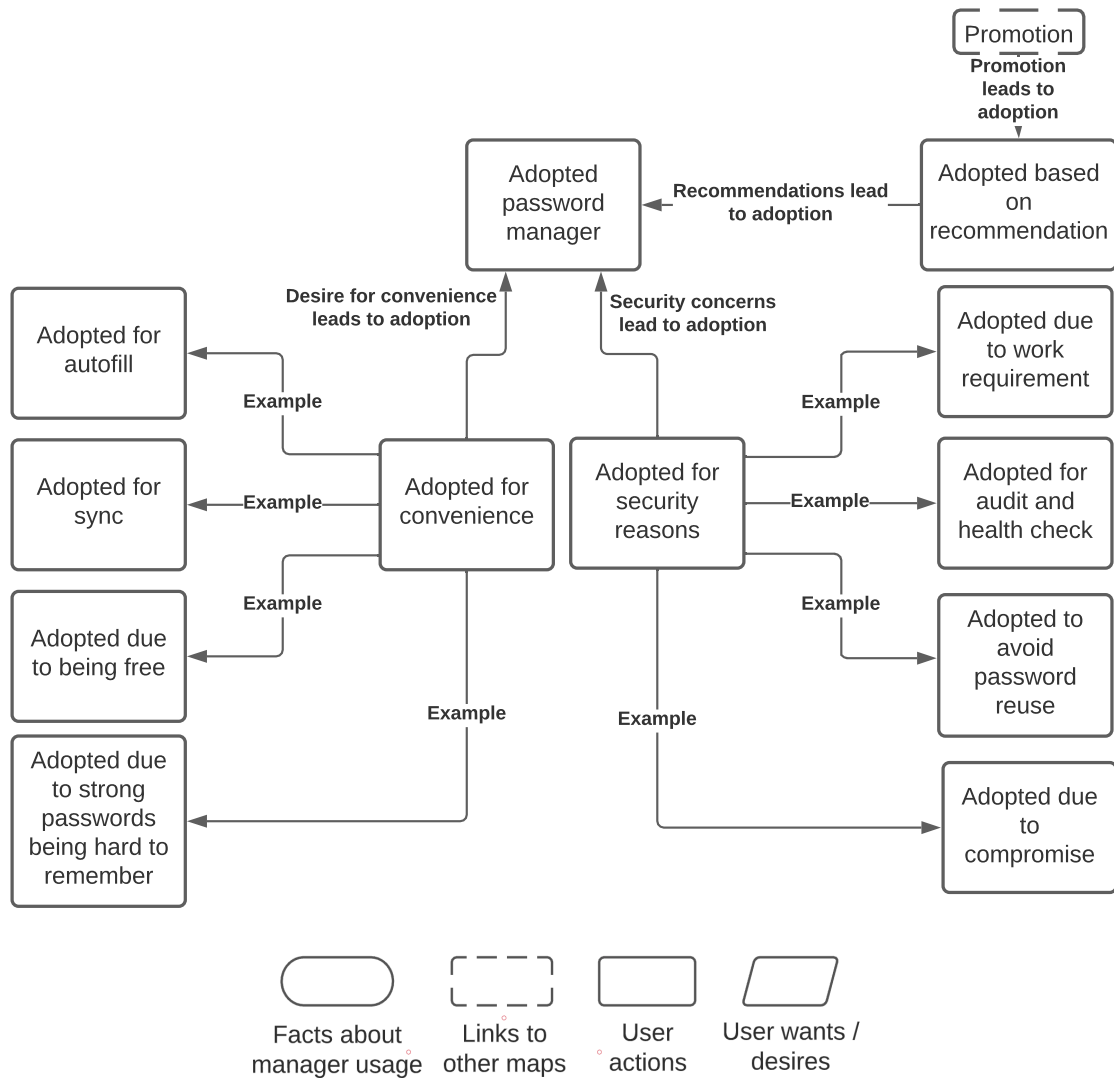
Several participants also shared that getting their parents to adopt a manager required the manager to be very easy to use, which is why they tended to recommend browser-based managers for their parents rather than more fully featured solutions. For example, R10 highlights that because keychain was so easy to use and did not require you to manage anything yourself, from syncing to generation, he was able to get his mother to adopt it.

R10: *“Yeah, so my mother. You know, who has, you know, uses either myself or my sister as technical support. She’s not gonna remember a complex password by any means. And her default is to use her cat’s name as a password. In this day and age, she just simply cannot do that for banking, or, you know, brokerage accounts or anything like that. It’s just, you can’t do it. So we showed her how to, you know, use keychain herself. And it pretty much, you know, takes care of anything you have to do for you, especially if you’re using it on your phone, and you could use face ID or whatever, like, you know, once you know how to use it, there’s no work involved, and she’s willing to do it. Whereas, for something more complicated, she wouldn’t be willing to.”*

5.3 Additional Topics

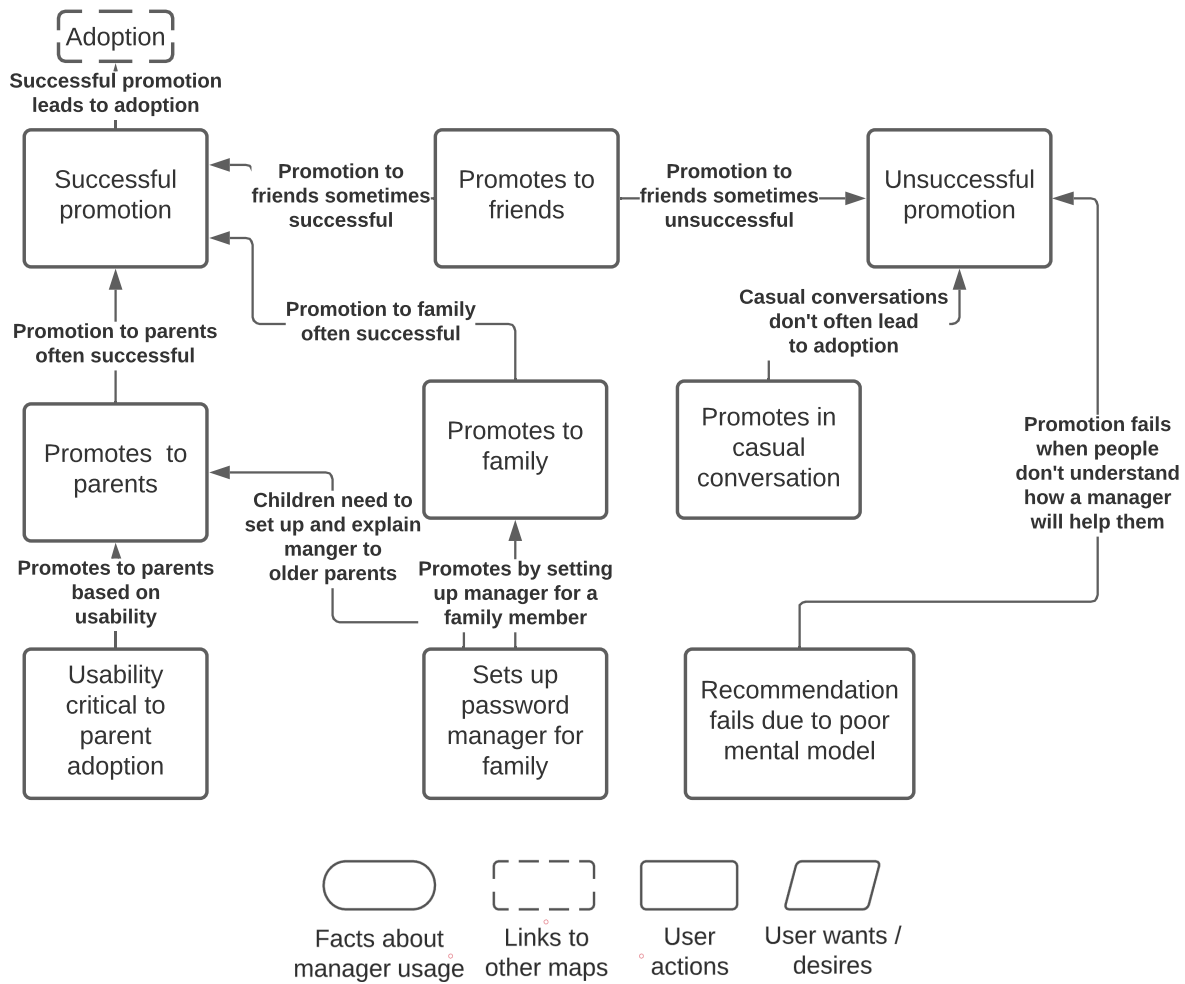
5.3.1 Password sharing via manager difficult and rarely used

Most users rarely need to share passwords and, when they do, it’s generally for unimportant accounts (e.g. Netflix) where they are not concerned about texting / emailing the password and with people who do not have the same or any password manager (see quotes from P3 and R26 below), so they could not use their manager’s sharing feature regardless. One participant had attempted to share a password with a friend via LastPass, but found it difficult to use even though he had a fairly good mental model of how it worked (see quote from R12



Participants who adopted managers on their own generally did so either for the sake of convenience or to keep their accounts more secure. When participants adopted a manager based on a recommendation, it generally came from a friend or family member or a requirement to adopt a manager at work.

Figure 5.6: Concept map for adoption.



Participants were most successful at promoting managers to immediate family members, parents, or close friends. Promotion in casual conversations was unsuccessful and several participants mentioned that promotion failed due to inaccurate mental models of how managers work.

Figure 5.7: Concept map for promotion.

below). Most users who shared important passwords, such as banking accounts, did so with their immediate family members (often with a spouse) and either shared the same manager account or used passwords that they both were familiar with to avoid the hassle of password sharing. One participant did mention that sharing via the manager might allow them to have a separate account from their spouse, which they felt might be helpful. Several participants also mentioned that this feature would be more useful at work, where they needed to share more sensitive accounts with other employees.

Doesn't know anyone who has a password manager - P3: *"Most people are too lazy to set up a password manager. So I'll just send an email or something."*

R26: *"Most of the people that have been in the scenario that you mentioned (need to share password), I don't believe are dashlane users. So it's probably a feature I've heard of, but I don't think it's a feature I've ever actually used. But I would."*

Password sharing via LastPass hard to use - R12: *"I did this (password sharing) with a friend. He was sharing with me using LastPass. And I figured with LastPass everything would be pretty normal. But no, it came through and wanted, like, oh, I had to accept it. And then he got a notification. And then there's some temporary password that just doesn't give me his actual password. It does some sort of encryption... It's kind of like those blue boxes, right, you know the the door locks for your house where you, you can authorize somebody to come in at one time and then you delete it right? That is the same thing. Except for like I said it didn't work. And then he shared it with me for a minute. And it popped in and out of six times. And every time I tried to use it, it would not work. And so he just ended up texting the password to me."*

5.3.2 Mental models of password reuse

Participants generally fell into three categories in terms of their mental model of password reuse: (1) They thought that having one very strong password (see R14 below) or a strong master password (see R1 below) made reuse okay, (2) They knew that reuse was bad and

either did not care or preferred convenience over security (see R9 below), or (3) They adopted a password manager specifically to stop reuse (see R19 below).

R1: *“10 years ago, when I was taking my security plus course, for the certification. They had a password cracker there. Well, I ran my normal passwords, and they got it pretty quick. And I ran my master password through. It said oh it would take 500,000 years to find that password. Then I said okay, I know what I’m doing now.”*

R14: *“I’ve been using them password that I have for 20 years. You know what I mean? It’s not, it’s not like my pet’s name or my dad or whatever, something like that. It’s like a, I don’t know, abstract, obscure, weirdo password that no be like, Oh, yeah, it is Richard Nixon. You know, I mean, like, it’s not, it’s not something that’s gonna be, you know, super related to me, it has nothing to do with something like that.”*

R9: *“Um, I have like, a few that I just kind of use in rotation. Which is probably not the best thing. But there we go.”*

R19: *“I was concerned about me just using a handful passwords, although they were fairly strong passwords, you know, they’re quite long. And we’re very, you know, like, sort of like just random strings of letters and numbers, like I was always trying to use long passwords. I only use like one or two of them. And I just felt like that wasn’t a good practice. So I wanted to get into a habit of using more.”*

5.3.3 Inconsistencies in autofill and autosave

While we already explained how mobile inconsistencies lead users to use managers less on their phones, we also found that users experience many inconsistencies with both autofill and autosave using managers on their desktop devices. R14 describes a common problem that occurred when the username field and password field are on separate pages, which confuses the manager and prevents autofill from functioning correctly. And P1 shares a common

concern with autosave where the password they generated is not saved and they did not store it, so they write it down to make sure they don't lose it if autosave fails. For both autosave and autofill, users often blamed either the website (see R12 below) or themselves (see R28) rather than the manager.

Inconsistencies in autofill - R14: *"Yes, sometimes like, I don't know how to explain it. It will do like a like, for example, my bank. Okay, you got to put your username first, then it offers it authorizes you, then you put in the password. And then it logs you in. Okay. It doesn't understand that there's two different steps there. Like so you need to put the username on page one. And the password on page two? Does it doesn't get that."*

Coping strategy when autosave fails - P1: *"I'll use the password generator, and I'll copy it from there into a notepad because I've had issues where the generated password doesn't go where I expect it to go. Or I even like lost it entirely. So I'll paste it over into Notepad, get my account created, make sure everything's sorted out in LastPass that the password and LastPass and the password I have in my notepad that I've copied and pasted over, make sure they match. And then I'll close the notepad if they do or fix any problems if there's a problem."*

Users blame website rather than manager for inconsistencies - R12: *"And so sometimes, if it's if it's not a super popular website, it won't understand that this is a password field, because there is no username field. There's certain websites that I actually have to just copy and paste the password because of that. And that's not really a fault of the password manager."*

User blames self for inconsistencies - R28: *"Um, I feel like once or twice, there's been maybe like a glitch in like updating something or it didn't. It didn't get the email down, right, probably due to user error on my part where I've gotten multiple accounts, and I might save a password for a different email to the wrong account. Maybe"*

5.3.4 Desire for single sign-on

Many users will use single sign-on on websites and especially in mobile apps whenever it is available rather than creating a separate account. In addition, several users, such as R25 below, expressed a clear desire for a mechanism like single sign-on to prevent the need to create and store so many passwords.

R25: *“They always think we can remember all these crazy password amounts. And like they tell you create a different password for every website. And I tried to do that and I found myself like locked out of all my websites. So if there’s like something that can be done to make it where the passwords can be simple, and we can reuse the passwords on different sites, but yet, they’re still strong. That would be like the best thing ever.”*

5.3.5 Ubiquitous usage of default settings

Only one of our participants actually changed a default setting when installing their manager on a new device and they could not find it during the interview. Even those participants who checked to make sure a setting was enabled, such as default timeout, did not actually change any settings (see R9 and P2 below). The name ‘Captain Default’, which R13 used to describe themselves, could aptly apply to basically all of our participants, at least in regard to their password manager usage.

R9: *“The settings? Yeah, usually I’ll just go through and just like, I’ll just check things. Usually I don’t really change anything. Honestly. Yeah, I just take it as it is.”*

P2: *“But honestly, I leave everything at the defaults. As far as functionality goes. I do check this default timeout on browser restart. I like that to be enabled. And it is by default as of right now. But yeah, really, I don’t change much of anything.”*

R13: *“No, we we call ourselves Captain default”*

5.3.6 Attitudes towards autolock

While more convenience oriented users felt having their manager lock itself periodically would only be an annoyance (see R14 below), even those more security oriented users did not feel autolock was necessary at home (see R11), or locked their device (laptop or phone) in order to lock their manager (see R10 and R7).

R14: *“That is a hindrance to somebody like me that works online all the time. I can’t stand it, say for example, I get up and go to the bathroom or get a drink or something. And then oh, my computer has gone to sleep. So I have to log in again? I’m sorry, I just don’t have time for that. If I’m here, you know, it should be on.”*

R11: *“I use it at work, I don’t use it at home. I used to use it work much more often when I was in the office. Now that I’m at home, I still use it. But it’s definitely not as prevalent. Because it is a good feature. It is good to have in case someone is walking by and they can see a password. But because I’m in my own house, and no one comes and sees me where I work in my home office. I don’t need to worry about that security as much. But in a crowded office, where I would normally be working. It’s very helpful.”*

R10: *“So I mean, in my house, who cares? It’s just me, but usually, if I’m not using it, you know, I shut my computer, at which point it is locked. ”*

R7: *“I do have my phone has is locked. So there’s that level of security?”*

5.3.7 How COVID-19 changed password manager usage

We encountered several participants who shared how the pandemic had altered their device usage or context and, as a result, their password manager usage. R28, who normally unlocked their phone using Face ID, was forced to enter their master password to unlock their mobile manager when wearing a mask because Face ID was not reliable. And R11, along with several other participants, mentioned that they did not use their phone as much because they were home more often, which meant they utilized their phone manager even less than normal. R11

also indicated that password sharing would be particularly useful during COVID because he could not share secure passwords via sticky note and subsequently destroy them via the confidential bin when everyone was working remotely.

R28: *“I need to login more frequently on my phone - it locks I believe after every use, but I can unlock it with my Face ID instead of having to enter the master password. These days (after COVID), though, I do still use the master password, because half the time I’m using my phone and I’ve got a mask down so it can’t actually tell me who I am.”*

R11: *“Occasionally (create accounts on phone), not as much as I used to (before COVID), but when I wasn’t always home. I would create accounts as well on my phone (more often). But I probably do it once a month at most now.”*

Interviewer: *“And if your password manager allowed you to share your passwords directly with another user of the same password manager securely, would that be a feature that you think you would use?”*

R11: *“Definitely - because emails can be intercepted. And depending on how high and the count is I’m trying to share with them, it would be easier that way. Back before the pandemic hit, I would just write it down on a post-it note, show them and then throw the post-it note in a confidential bin. We can’t do that anymore. Because I don’t know. Everyone’s in a different part of the state.”*

5.4 Discussion

In this section we highlight several areas where our work provides new insights into or differs from the results of prior work, and then discuss ways to address the issues we identified in our study. Lyastani et al. [41] found that password reuse still occurred even among manager users, and that the key to stronger passwords was using the manager for the entire password lifecycle (creation, storage, autofill). In our research, we identified multiple reasons that users prefer not to use their manager for their entire lifecycle that need to be addressed. In particular, users considered generated passwords hard to enter cross-device and hard to remember in

the event that their manager was not available, which resulted in them avoiding generation. Users were also forced to enter passwords manually due to problems with syncing across devices and experienced inconsistencies with both autofill and autosave, which encouraged the use of easier to remember passwords rather than generation.

Our results also help explain why users of browser-based managers are generally more convenience oriented and users of third-party managers more concerned with security. Users often adopt the browser manager without any forethought. As noted by one of our users, "it starts using them". In contrast, users of third-party managers usually adopt them based on recommendations (either online or from people they know) or because they are required to do so by their work. It makes sense that people who adopt a manager at work or based on a recommendation would have a better mental model of how their manager works than those who adopt it simply because it pops up. Another potential explanation is that only people who are more security concerned are willing to through the effort to set up a third-party manager.

Seiler Hwang et al. [56] reported that users changed at least some of the default security settings in most cases. However, we found that nearly all of our participants used default settings across the board. We believe this difference may be due to study design. They prompted participants to complete the task of changing settings, which may have made them feel they should change at least some settings. In contrast, we observed participants and asked them to show us what settings they would normally change, if anything, and most responded that they did not change any settings.

5.4.1 Reducing Multiple Manager Usage.

Users utilize multiple managers at once, but password managers are not designed to support this use case. There are two potential responses to multiple manager usage: (1) Prevent multiple manager usage by addressing the underlying issues that cause it or (2) Design managers to cooperate better when used together. While it is unlikely that all issues causing multiple manager usage (e.g. persistence of browser-based managers, unreliable syncing, fears of not having manager available or the manager failing) will be addressed in the near future, we believe that the best approach is to nudge users away from multiple manager usage.

In addition to the difficulties of convincing password manager vendors to support multiple manager usage, from a security standpoint it is detrimental. Browser-based managers are typically less secure and fully featured than third-party managers [48] and storing passwords in two locations leaves users more vulnerable to attack. We recommend the following practices to reduce multiple manager usage:

(1) Third-party managers should walk users through the process of exporting credentials out of their browser manager and then disabling the browser manager. Managers such as LastPass can import credentials directly from the browser, but none of the participants in our study used this feature. We also found that some users would sync credentials to their third-party manager by first autofilling with the browser manager and then autosaving into their third-party manager. And none of our participants mentioned disabling the browser manager, even though some of them avoided the browser manager by constantly choosing not to autosave their credentials. These results suggest that users are unaware of these features. (2) More focus on making syncing reliable and helping users configure their manager across their devices. Some of our participants were not even aware that their manager offered a mobile solution or that they could sync between their devices simply by using the same account, even if they had used the manager for five years or more. Periodic pop-ups reminding users that they can sync across devices with instructions on how to do so if they want to take advantage of that future may be helpful in this regard. (3) Alleviating users fears that they may lose their passwords and offering instructions on how to back up their vault if they have this concern. Giving users clear instructions on how to back up their data would hopefully prevent them from resorting to multiple manager usage for a sense of security.

5.4.2 Addressing Concerns with Generated Passwords

Users often avoid using the password generator because generated passwords are hard to remember when their manager is not available and hard to enter cross-device. The result is that users resort to password reuse or creating weak passwords. Generators need to implement modes to address ease of entry and memorability and they need to ensure that users are aware of these features. Prior research has explored ways to make passwords easier together on alternate devices [34] and more research is needed to find the optimal way to implement

cross-device password generation. However, we suspect that most users would consider an easy to remember password good enough for cross-device usage, even if it is not optimized for entry. As long as managers help users understand that they can generate easy to remember passwords and this feature is easy to access, we believe that alone will encourage more users to adopt the password generator.

5.4.3 Health Check Should Be Seamless and Targeted.

Even the most advanced users main use case for their manager is to remember and fill their passwords. The benefits of features such as health check are not users' priority and they are unlikely to navigate to a separate page specifically for that feature unless they already self audit their accounts. However, users are willing to respond to pop-up breach notifications about accounts that they consider important. We therefore recommend that managers do the following to increase the usability of health check:

(1) Use a pop-up to notify users when an account is breached or passwords are weak, but give users the option to disable or change the frequency of these notifications via the same pop-up. This operation should feel seamless to the user—almost as if it is automated. (2) Develop an algorithm to identify which accounts are important to users or allow users to indicate which accounts they would be willing to update. A pop-up asking users which types of accounts they consider important (e.g. banking, ecommerce) may be all that is needed. Future research should investigate the optimal way to offer more targeted health check results to users.

Chapter 6

Conclusion and Future Work

In this dissertation, I have analyzed the security of password managers across the entire password lifecycle and contributed to the research community’s understanding of how and why users use managers across their devices. I began with a security evaluation of generation, storage, and autofill on the desktop (Chapter 3). We replicated prior work that identified security concerns with autofill in the browser, as well conducting the first evaluation of generation using a corpus of 147 million generated passwords. Our work found that while managers had improved since prior work, there were still security concerns, especially related to autofill in the browser.

Moving from desktop to mobile, in Chapter 4 I presented an analysis of mobile autofill frameworks on iOS and Android. These frameworks are intended to provide a secure mechanism for password managers to autofill credentials, but we found that in some cases they actually enforced poor behaviors (such as filling cross-origin iframes on iOS) or left too much room for managers to make implementation errors (such as app-to-domain mapping on Android). We recommended concrete steps to make these frameworks more secure and noted several framework smells that point towards potential problems with an autofill framework (similar to how code smells point to problems with code).

Finally, in Chapter 5 I discussed the results of a semi-structured observational interview of password manager users. The core result of this work was five theories about how users utilize managers: multiple manager usage, cross-device entry and reuse, limited usage on mobile, health check desired but overwhelming, and patterns in adoption and promotion.

Multiple manager usage was a phenomena never discussed by prior work and we found it to be a common pattern of usage (most often a browser and external manager used in tandem). We also found that users tend to avoid the password generator because generated passwords are difficult to enter cross-device or when their manager is not available. As noted by prior work [41], users who avoid the generator are more likely to reuse or create weak passwords.

In addition, we found that users who adopt browser managers often do so simply because the manager pops up, while those who adopt third-party managers (e.g. LastPass, 1Password, Dashlane) often do so due to requirements at work or based on recommendations. This pattern in adoption helps explain why users of browser managers are more convenience oriented and lack as clear a mental model of how managers work as users of third-party managers [72]. Other results included theories related to adoption/promotion and limited usage on mobile, as well as password sharing, mental models of reuse, inconsistencies in autofill and autosave, attitudes towards autolock and manager security, and effects of COVID-19 on manager usage.

Below I will discuss key lessons learned while conducting this research and how future research can begin to address outstanding issues.

6.1 Lessons Learned

Reflecting on the results of my analysis of password manager security and usability, six major themes stand out:

6.1.1 Managers need better support.

Managers need better support from and integration with browsers, applications, and OSes. Many of the security and usability challenges faced by manager users are not actually the fault of the manager. The problem is that managers are working in an environment where they are not first class citizens and functionality such as password generation and secure autofill is not fully supported. On mobile devices, the problem is exacerbated by the fact that managers have to work in multiple contexts (apps, browser, WebView) and cooperate with mobile autofill frameworks that have their own flaws.

6.1.2 Inconsistencies lead to reuse.

Inconsistent functionality (e.g. autofill, autosave, syncing) leads to password reuse and other coping strategies. When users experience inconsistencies in manager behavior, they resort to known strategies for password creation and storage, sometimes even abandoning features meant to provide convenience and security. For example, when syncing across devices is not intuitive users choose passwords that are easier to remember so that they don't have to look them up on their desktop when using their mobile device. Or when autosave and autofill are inconsistent, users resort to manually entering passwords into their manager initially and then copying and pasting them into form fields. Consistency is critical to prevent users from falling back on known password creation strategies or using coping mechanisms as their primary method to complete common tasks.

6.1.3 Usability issues prevent full adoption.

Usability hurdles prevent many users from adopting managers for the entire password lifecycle. For example, because generated passwords are hard to enter manually and hard to remember, some users avoid using the generator altogether. They would rather use weaker passwords than be caught without their password on a device where their manager is not installed or have to enter a complex, random password on their TV. To ensure users adopt managers for the entire password lifecycle, usability across that lifecycle, including edge cases, must be considered. Users will avoid a feature in order to avoid edge cases.

6.1.4 User concerns prevent full adoption.

Managers need to address user concerns such as the fear of losing passwords and non-availability. Related to the prior point about usability and edge cases, managers need to address the very real concerns of users. Users are concerned about putting all of their passwords in one place. They are worried that if their manager fails and all of their passwords are long and complicated they will be locked out of all of their accounts. And they are afraid that they might lose their passwords. While these fears are not pervasive among users, they need to be more directly addressed to encourage users to make full use of their manager.

6.1.5 Inaccurate mental models are still an issue.

Incorrect mental models prevent users from experiencing the full benefit of their manager or even adopting a manager in the first place. Even users who are concerned about security may lack a clear mental model of how their manager works. For example, we had one security concerned participant who was a retired IT worker and thought that a strong master password meant their other passwords did not need to be strong. In addition, we found that participants who promoted their managers encountered non-users who simply did not understand why reuse was bad or how a manager would benefit them. There is a need to find new ways to educate manager users and engage the broader public with information on how to stay secure online.

6.1.6 Secure autofill should be a priority.

Implementing secure autofill should be a priority for both managers and researchers. Autofill is a key security concern for managers both on the desktop and on mobile devices, even though there are known methods to make it secure [60]. Several attempts have been made to implement a secure autofill mechanism, but they have all failed to find widespread adoption. Finding a way to implement secure autofill across devices that browsers, mobile operating systems, apps, and vendors can all adopt should be a priority for the password manager community.

6.2 Future Work

Based on the results of my research, I recommend the following avenues for future research:

6.2.1 Browser-Supported Password Managers.

Currently, authentication is a second-class citizen within browsers. Future research should examine how browsers can better support password-based authentication—for example, making password-based authentication interfaces first-class HTML elements that the browser implements to ensure that passwords are handled correctly. This could include providing

a common, recognizable interface for password-based authentication, allowing for the use of alternative protocols (e.g., strong password protocols [7, 69]), and preventing malicious websites from creating look-alike phishing interfaces [54].

Research should also explore how browsers can provide additional features to password manager extensions. Examples include, (1) allowing password managers to generate a nonce to autofill in place of the password that the browser will replace with the password when it is transmitted to the website if and only if the target domain matches the domain associated with the password in the password manager [60] (see Section 3.4.5); (2) providing password managers access to the system keyring (e.g., macOS keyring, Windows Vault), giving them a more secure and standardized mechanism for storing account credentials; (3) handling the user interaction component of autofill and ensuring that it is clickjack resilient; (4) adding HTML attributes that describe a website’s password policy, allowing password managers to generate passwords that will be accepted by the website [59]. It would also be worth examining whether such annotations could be automatically inferred and added by semantically evaluating the code that checks passwords.

6.2.2 Helping Users with Cross-device Entry

More research is needed to understand how managers can best help users enter passwords on devices where their manager is not available. We found that users often resort to weak or reused passwords for cross-device entry because generated passwords are difficult to enter manually and hard to remember. One approach discussed in Chapter 5 is to add an “easy to remember” mode to generators and encourage users to use it when creating passwords they will need to use without their manager or enter on alternate devices (e.g. TV, Xbox). But a user study is required to determine if users would actually utilize this feature and if it would alleviate their concerns.

Another possible solution would be to allow users to log in to other devices using their phone as a portable manager. The downside to this approach is that many users do not use their manager on their phone, so this solution would require broader adoption of mobile managers. In addition, users would need to understand yet another feature of their manager when many users already lack a clear mental model of how their manager works.

6.2.3 Secure Credential Mapping.

We believe that research needs to be conducted to design a mechanism that allows a domain to indicate which webpages should be allowed to receive autofilled credentials. This would prevent vulnerable webpages on the domain other than these login pages from being able to steal users autofilled credentials, especially if password managers prevent autofill within same-origin iframes. We believe this feature could be implemented similarly to how mappings work for iOS Password AutoFill or DAL files, having a single file on the website that lists acceptable URLs. Still, research is needed to identify the feasibility and effectiveness of this proposal, along with the best way to implement it.

Research is also needed to protect against attacks that utilize WebView on mobile devices. A secure mapping between credentials and websites should be enforced in the context of a WebView and the app hosting the WebView should not be able to inject JavaScript into a random website without appropriate permissions. In the case that an app is hosting their own domain inside the WebView and JavaScript injection is desired to enhance the user experience, a secure mapping between website and app should be required.

6.2.4 Research-Derived Character Sets.

Password managers generate passwords using different character sets, differing dramatically in which symbols they allow and which characters they remove as unusable (e.g., difficult to remember, hard to distinguish). We advocate for a data-driven effort to establish standardized character sets.

User studies should be conducted to identify the characters that are difficult for users to read and input, with attention paid to alternative input modalities (e.g., entering passwords using a TV remote or accessible keyboard). Measurements of existing password policies could also be used to identify which characters are commonly rejected by website password policies. It may be that there is no one ideal character set, but rather different character sets for different types of passwords (e.g., passwords with restrictive policies, passwords entered with non-keyboard modalities). In this case, statistical modeling could be used to identify the ideal lengths for passwords in various modalities.

6.2.5 Autofill Framework for the Desktop.

Research is needed on creating an autofill framework for desktop environments. While browsers do provide a platform to deploy password manager extensions, they do not actually provide any password management-centric functionality—i.e., they do not assist with the detection of login forms nor facilitate autofilling credentials. This lack of framework support causes a mixed level of security for password manager extensions. Moreover, there is no OS-level autofill framework, making it nearly impossible for passwords managers to provide universal autofill for desktop applications.

Bibliography

- [1] Alkaldi, N. and Renaud, K. (2016). Why do people adopt, or reject, smartphone password managers? *EuroUSEC*. 10, 12
- [2] Anderson, C. L. and Agarwal, R. (2010). Practicing safe computing: A multimethod empirical examination of home computer user security behavioral intentions. *MIS quarterly*, pages 613–643. 101
- [3] Aonzo, S., Merlo, A., Tavella, G., and Fratantonio, Y. (2018). Phishing attacks on modern android. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1788–1801. 8, 13, 51, 52, 156
- [4] Arias-Cabarcos, P., Marín, A., Palacios, D., Almenárez, F., and Díaz-Sánchez, D. (2016). Comparing password management software: toward usable and secure enterprise authentication. *IT Professional*, 18(5):34–40. 12
- [5] Aurigemma, S., Mattson, T., and Leonard, L. (2017). So much promise, so little use: What is stopping home end-users from using password manager applications? 10
- [6] Bakry, T. H. and Mysk, T. (2020). Precise location information leaking through system pasteboard. <https://www.mysk.blog/2020/02/24/precise-location-information-leaking-through-system-pasteboard/>. Accessed: 2020-06-13. 9, 146
- [7] Bellare, S. and Merritt, M. (1992). Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Proceedings of the 1992 IEEE Symposium on Research in Security and Privacy*, pages 72–84. IEEE. 132
- [8] Bianchi, A., Corbetta, J., Invernizzi, L., Fratantonio, Y., Kruegel, C., and Vigna, G. (2015). What the app is that? deception and countermeasures in the android user interface. In *2015 IEEE Symposium on Security and Privacy*, pages 931–948. IEEE. 9
- [9] Bojinov, H., Bursztein, E., Boyen, X., and Boneh, D. (2010). Kamouflage: Loss-resistant password management. In *European symposium on research in computer security*, pages 286–302. Springer. 7

- [10] Bonneau, J. (2012). The science of guessing: analyzing an anonymized corpus of 70 million passwords. In *2012 IEEE Symposium on Security and Privacy*, pages 538–552. IEEE. 23
- [11] Bonneau, J., Herley, C., Van Oorschot, P. C., and Stajano, F. (2012). The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *2012 IEEE Symposium on Security and Privacy*, pages 553–567. IEEE. 1
- [12] Bravo-Lillo, C., Cranor, L., Downs, J., Komanduri, S., Schechter, S., and Sleeper, M. (2012). Operating system framed in case of mistaken identity: measuring the success of web-based spoofing attacks on os password-entry dialogs. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 365–377. ACM. 16
- [13] Chatterjee, R., Bonneau, J., Juels, A., and Ristenpart, T. (2015). Cracking-resistant password vaults using natural language encoders. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 481–498. IEEE. 7
- [14] Chiasson, S., van Oorschot, P. C., and Biddle, R. (2006). A usability study and critique of two password managers. In *USENIX Security Symposium*, volume 15, pages 1–16. 10, 11, 12
- [15] Choong, Y.-Y. (2014). A cognitive-behavioral framework of user password management lifecycle. In *International Conference on Human Aspects of Information Security, Privacy, and Trust*, pages 127–137. Springer. 6, 12, 14, 15
- [16] Chromium (2019). Linux password storage. https://chromium.googlesource.com/chromium/src/+master/docs/linux_password_storage.md. Accessed: 2019-05-20. 35
- [17] Das, A., Bonneau, J., Caesar, M., Borisov, N., and Wang, X. (2014). The tangled web of password reuse. In *NDSS*, volume 14, pages 23–26. 1
- [18] Dell’Amico, M., Michiardi, P., and Roudier, Y. (2010). Password strength: An empirical analysis. In *2010 Proceedings IEEE INFOCOM*, pages 1–9. IEEE. 1

- [19] Dhamija, R., Tygar, J. D., and Hearst, M. (2006). Why phishing works. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 581–590. 8, 51, 74, 76, 89
- [20] Egelman, S. and Peer, E. (2015). Scaling the security wall: Developing a security behavior intentions scale (sebis). In *Proceedings of the 33rd annual ACM conference on human factors in computing systems*, pages 2873–2882. 95, 96
- [21] Evaluators, I. S. (2019). Password managers: Under the hood of secrets management. <https://www.securityevaluators.com/casestudies/password-manager-hacking/>. Accessed: 2019-02-22. 1, 7, 18, 19
- [22] Fagan, M., Albayram, Y., Khan, M. M. H., and Buck, R. (2017). An investigation into users’ considerations towards using password managers. *Human-centric Computing and Information Sciences*, 7(1):12. 11
- [23] Fahl, S., Harbach, M., Oltrogge, M., Muders, T., and Smith, M. (2013). Hey, you, get off of my clipboard. In *International Conference on Financial Cryptography and Data Security*, pages 144–161. Springer. 9, 148
- [24] Felt, A. P. and Wagner, D. (2011). *Phishing on mobile devices*. na. 8, 51, 74, 76, 89
- [25] Fernandes, E., Chen, Q. A., Paupore, J., Essl, G., Halderman, J. A., Mao, Z. M., and Prakash, A. (2016). Android ui deception revisited: Attacks and defenses. In *International Conference on Financial Cryptography and Data Security*, pages 41–59. Springer. 9
- [26] Florencio, D. and Herley, C. (2007). A large-scale study of web password habits. In *Proceedings of the 16th international conference on World Wide Web*, pages 657–666. ACM. 1
- [27] Florêncio, D., Herley, C., and Van Oorschot, P. C. (2014). An administrator’s guide to internet password research. In *28th Large Installation System Administration Conference (LISA14)*, pages 44–61. 24

- [28] Fratantonio, Y., Qian, C., Chung, S. P., and Lee, W. (2017). Cloak and dagger: from two permissions to complete control of the ui feedback loop. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 1041–1057. IEEE. 10, 53, 149
- [29] Gasti, P. and Rasmussen, K. B. (2012). On the security of password manager database formats. In *European Symposium on Research in Computer Security*, pages 770–787. Springer. 1, 2, 7, 12, 14, 15, 36, 44
- [30] Goel, D. and Jain, A. K. (2018). Mobile phishing attacks and defence mechanisms: State of art and open research challenges. *Computers & Security*, 73:519–544. 9
- [31] Grassi, P., Fenton, J., Newton, E., Perlner, R., Regenscheid, A., Burr, W., Richer, J., Lefkovitz, N., Danker, J., Choong, Y.-Y., Greene, K., and Theofanos, M. (2020). Nist special publication 800-6b: Digital identity guidelines. *National Institute of Standards and Technology*. 82
- [32] Green, M. (2014). Why can’t apple decrypt your iPhone? <https://blog.cryptographyengineering.com/2014/10/04/why-cant-apple-decrypt-your-iphone/>. Accessed: 2020-06-16. 82
- [33] Greene, K. K. (2015). Effects of password permutation on subjective usability across platforms. In *International Conference on Human Aspects of Information Security, Privacy, and Trust*, pages 59–70. Springer. 79
- [34] Greene, K. K., Gallagher, M. A., Stanton, B. C., and Lee, P. Y. (2014). I can’t type that! p@ \$\$w0rd entry on mobile devices. In *International Conference on Human Aspects of Information Security, Privacy, and Trust*, pages 160–171. Springer. 126
- [35] Herley, C. (2009). So long, and no thanks for the externalities: the rational rejection of security advice by users. In *Proceedings of the 2009 workshop on New security paradigms workshop*, pages 133–144. ACM. 1
- [36] Jang, Y., Song, C., Chung, S. P., Wang, T., and Lee, W. (2014). A11y attacks: Exploiting accessibility in operating systems. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 103–115. 10, 53, 149

- [37] Kang, R., Brown, S., Dabbish, L., and Kiesler, S. (2014). Privacy attitudes of mechanical turk workers and the us public. In *10th Symposium On Usable Privacy and Security ({SOUPS} 2014)*, pages 37–49. [101](#)
- [38] Lee, J.-W. and Kim, I.-S. (2016). A study on the vulnerability of security keypads in android mobile using accessibility features. *Journal of the Korea Institute of Information Security and Cryptology*, 26(1):177–185. [10](#), [53](#), [149](#)
- [39] Li, Z., He, W., Akhawe, D., and Song, D. (2014). The emperor’s new password manager: Security analysis of web-based password managers. In *USENIX Security Symposium*, pages 465–479. [1](#), [2](#), [5](#), [7](#), [12](#), [14](#), [15](#), [37](#), [43](#), [44](#), [48](#), [51](#)
- [40] Luo, T., Jin, X., Ananthanarayanan, A., and Du, W. (2012). Touchjacking attacks on web in android, ios, and windows phone. In *International Symposium on Foundations and Practice of Security*, pages 227–243. Springer. [9](#), [10](#), [53](#), [74](#), [76](#), [89](#)
- [41] Lyastani, S. G., Schilling, M., Fahl, S., Backes, M., and Bugiel, S. (2018). Better managed than memorized? studying the impact of managers on password strength and reuse. In *27th USENIX Security Symposium*, pages 203–220. [1](#), [3](#), [6](#), [10](#), [11](#), [13](#), [94](#), [124](#), [129](#)
- [42] Masur, P. K. (2018). *Situational privacy and self-disclosure: Communication processes in online environments*. Springer. [95](#)
- [43] Melicher, W., Kurilova, D., Segreti, S. M., Kalvani, P., Shay, R., Ur, B., Bauer, L., Christin, N., Cranor, L. F., and Mazurek, M. L. (2016a). Usability and security of text passwords on mobile devices. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 527–539. [79](#)
- [44] Melicher, W., Ur, B., Segreti, S. M., Komanduri, S., Bauer, L., Christin, N., and Cranor, L. F. (2016b). Fast, lean, and accurate: Modeling password guessability using neural networks. In *25th USENIX Security Symposium*, pages 175–191. [23](#), [24](#)
- [45] Mozilla (2019). The autocomplete attribute and login fields. https://developer.mozilla.org/en-US/docs/Web/Security/Securing_your_site/Turning_off_form_

- [autocomplete#The_autocomplete_attribute_and_login_fields](#). Accessed: 2019-11-12. 43
- [46] Naseri, M., Borges, N. P., Zeller, A., and Rouvoy, R. (2019). Accessileaks: Investigating privacy leaks exposed by the android accessibility service. *Proceedings on Privacy Enhancing Technologies*, 2019(2):291–305. 10, 53, 149
- [47] Network, C. S. (2020). Is hotel wifi safe? no, and here’s why. <https://cybercrimesupport.org/is-hotel-wifi-safe/>. Accessed: 2020-10-08. 51
- [48] Oesch, S. and Ruoti, S. (2020). That was then, this is now: A security evaluation of password generation, storage, and autofill in browser-based password managers. In *29th USENIX Security Symposium (USENIX Security 20)*, Boston, MA. USENIX Association. 48, 51, 52, 58, 60, 61, 63, 78, 79, 83, 126, 152
- [49] O’Neill, M., Ruoti, S., Seamons, K., and Zappala, D. (2016). Tls proxies: Friend or foe? In *Proceedings of the 2016 Internet Measurement Conference*, pages 551–557. ACM. 39
- [50] Pearman, S., Thomas, J., Naeini, P. E., Habib, H., Bauer, L., Christin, N., Cranor, L. F., Egelman, S., and Forget, A. (2017). Let’s go in for a closer look: Observing passwords in their natural habitat. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 295–310. ACM. 1, 13
- [51] Phellas, C. N., Bloch, A., and Seale, C. (2011). Structured methods: interviews, questionnaires and observation. *Researching society and culture*, 3:181–205. 101
- [52] Ray, H., Wolf, F., Kuber, R., and Aviv, A. J. (2020). Why older adults (don’t) use password managers. *arXiv preprint arXiv:2010.01973*. 11
- [53] Riley, S. (2006). Password security: What users know and what they actually do. *Usability News*, 8(1):2833–2836. 1
- [54] Ruoti, S. and Seamons, K. (2017). End-to-end passwords. In *Proceedings of the 2017 New Security Paradigms Workshop*, pages 107–121. ACM. 132

- [55] Scorecard, S. (2018). Statistics: Cybersecurity data breaches on the rise. <https://securityscorecard.com/blog/cybersecurity-data-breaches-statistics-on-the-rise>. Accessed: 2019-02-22. 1
- [56] Seiler-Hwang, S., Arias-Cabarcos, P., Marin, A., Almenares, F., Diaz-Sanchez, D., and Becker, C. (2019). "i don't see why i would ever want to use it" analyzing the usability of popular smartphone password managers. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1937–1953. 10, 12, 125
- [57] Shahriar, H., Klintic, T., Clincy, V., et al. (2015). Mobile phishing attacks and mitigation techniques. *Journal of Information Security*, 6(03):206. 9
- [58] Silver, D., Jana, S., Boneh, D., Chen, E. Y., and Jackson, C. (2014). Password managers: Attacks and defenses. In *USENIX Security Symposium*, pages 449–464. 1, 2, 7, 12, 14, 15, 37, 40, 43, 44, 45, 48, 51, 63
- [59] Stajano, F., Spencer, M., Jenkinson, G., and Stafford-Fraser, Q. (2014). Password-manager friendly (pmf): Semantic annotations to improve the effectiveness of password managers. In *International Conference on Passwords*, pages 61–73. Springer. 132
- [60] Stock, B. and Johns, M. (2014). Protecting users against xss-based password manager abuse. In *Proceedings of the 9th ACM symposium on Information, computer and communications security*, pages 183–194. ACM. 1, 2, 7, 12, 14, 15, 37, 40, 42, 43, 44, 45, 48, 51, 52, 62, 63, 86, 88, 131, 132
- [61] Tan, J., Bauer, L., Bonneau, J., Cranor, L. F., Thomas, J., and Ur, B. (2017). Can unicorns help users compare crypto key fingerprints? In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 3787–3798. 100
- [62] Technologies, P. (2020). Web applications vulnerabilities and threats: statistics for 2019. <https://www.ptsecurity.com/ww-en/analytics/web-vulnerabilities-2020/>. Accessed: 2020-10-08. 51

- [63] Tuncay, G. S., Demetriou, S., and Gunter, C. A. (2016). Draco: A system for uniform and fine-grained access control for web code on android. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 104–115. 10
- [64] Tuncay, G. S., Qian, J., and Gunter, C. A. (2020). See no evil: Phishing for permissions with false transparency. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, pages 415–432. 9, 74, 76, 89
- [65] W3C (2019). Html. <https://www.w3.org/TR/html52/sec-forms.html#element-attrdef-autocompleteelements-autocomplete>. Accessed: 2019-11-09. 43
- [66] Wang, K. C. and Reiter, M. K. (2018). How to end password reuse on the web. *arXiv preprint arXiv:1805.00566*. 1
- [67] Wheeler, D. L. (2016). zxcvbn: Low-budget password strength estimation. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 157–173. 23, 24
- [68] Wu, L., Du, X., and Wu, J. (2014). Mobifish: A lightweight anti-phishing scheme for mobile phones. In *2014 23rd International Conference on Computer Communication and Networks (ICCCN)*, pages 1–8. IEEE. 9
- [69] Wu, T. et al. (1998). The secure remote password protocol. In *Internet Society Symposium on Network and Distributed System Security*. 132
- [70] Yang, G., Huang, J., and Gu, G. (2019). Iframes/popups are dangerous in mobile webview: studying and mitigating differential context vulnerabilities. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 977–994. 10, 53
- [71] ZDNet (2020). Fbi warns about ongoing attacks against software supply chain companies. <https://www.zdnet.com/article/fbi-warns-about-ongoing-attacks-against-software-supply-chain-companies/>. Accessed: 2020-10-08. 51

- [72] Zhang, S. A., Pearman, S., Bauer, L., and Christin, N. (2019). Why people (don’t) use password managers effectively. In *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*. [3](#), [10](#), [11](#), [94](#), [129](#)
- [73] Zhang, X. and Du, W. (2014). Attacks on android clipboard. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 72–91. Springer. [9](#)

Appendices

A Evolution of Mobile Autofill

In this section, we describe the evolution of autofill frameworks on iOS and Android. In our work, we evaluated the iOS app extensions and Password AutoFill frameworks as well as the Android autofill service.

A.1 Autofill on iOS

There are five ways autofill is handled in iOS: copy and paste, in-app browsers, autofill in browsers using browser-based password managers, using app extensions (introduced in iOS 8) [2014], and using the iOS Password Autofill framework (introduced in iOS 12 [2018]).

Copy and Paste

Prior to iOS 8, non-browser based password managers required the user to first open the password manager, select the appropriate credential, copy the username and password (separately), switch to the target app, and finally paste their credentials. Not only is this a poor usability experience, but this leaves users credentials at risk because other apps can also access the credentials stored in the clipboard. While iOS limits clipboard access to foreground apps, there is still the possibility that a malicious app opened in the future could access a previously copied password. This can be made even more likely by having the malicious app be a widget displayed on the users “Today” screen [6].

In-App Browser

One early approach taken by mobile passwords managers was to implement a fully functional browser as part of the password manager app. While this approach provides the most accurate autofill experience, it is less than unusable in that it does not help users authenticate to the numerous other applications on the mobile device. While some password managers continue to provide this functionality, it is not the preferred method for autofill anymore.

Browser-Based Password Manager

Many browsers on iOS provide their own password manager, which can then be used to autofill passwords into websites displayed through the browser. While this does work even if app extensions or Password AutoFill are unavailable, this approach is limited in that it cannot be used to autofill credentials into other apps.

App Extensions

App extensions allow a host app to interact with another app (e.g., a password manager) using a predefined set of extension features. This approach requires both the password manager to implement the set of functions associated with the password management extension feature and for host apps to be updated to query this extension feature. Using app extensions is more secure and usable than the copy and paste approach, though it does have two important limitations. First, host apps are free to request passwords for any domain and it is up to the user to ensure that only correct credentials are selected. Second, app extensions require the modification of individual host apps, an approach that does not scale well. Figure 4.1 shows the interface for app extensions, first requiring the user to select which app extension to use (see Figure 4.1a) and the selecting the credentials (see Figure 4.1b).

While Password AutoFill is intended to replace using app extensions for password management, we note that the password management extension feature is still supported in iOS and remains functional in some password managers (e.g., 1Password, Keeper, LastPass) and host apps (e.g., Safari, Edge). Additionally, for older devices that cannot be updated to iOS 12, app extensions remain the preferred method for password autofill.

Password AutoFill

The Password AutoFill framework provides two major benefits compared to app extensions. First, it automatically identifies login forms in apps and websites. It is preferred that apps add appropriate `textContentType` attributes to form fields to ensure correct login form detection, but detection will still proceed using a heuristic-based approach if these attributes are not present. Second, Password AutoFill allows for a secure mapping between an app

and the domains that should have their credentials entered into that app. That is done by having app developers include an Associated Domains Entitlement that indicates which domains are associated with the app; the domain operator is also required to include an `apple-app-site-association` file on their website indicating which apps are allowed to use credentials for that domain. Figure 4.2 shows the interface for Password AutoFill, both when an associated domain can be found (see Figure 4.2a) and when not (see Figure 4.2b).

A.2 Autofill on Android

While the evolution of autofill on Android is similar to that of iOS in its early stages, it diverged during the development of a formal framework. Android took a less strict approach in enforcing correct behavior than iOS, as described in detail below.

Copy and Paste

In contrast to iOS, Android does allow clipboard access to background apps [23], which can leave credentials vulnerable to theft unless password managers clear the clipboard.

Accessibility Service

The first approach that allowed filling passwords into other applications was to leverage the accessibility service provided by Android. While the purpose of the accessibility service is to help users with disabilities, it has several features that allow it to be used to implement filling passwords (though the developer recommendations do advise against using the accessibility service for non-accessibility purposes). First, it allows an accessibility app—the password manager in this case—to scan the visual elements being displayed in the current app; this is used by the password manager to identify login forms. Second, it allows the accessibility app to overlay additional interfaces over the current applications; this is used by the password manager to have the user select which account credentials to fill. Third, it allows the accessibility app to enter text for the user; this is used by the password manager to actually fill the user’s credentials. One drawback of this approach is it does require the users to give accessibility permissions to the password manager, which is not a straightforward process.

Additionally, research has shown that relying on the accessibility service introduces numerous security vulnerabilities [28, 36, 38, 46].

Android Autofill Service

In 2017, Android introduced the autofill service as part of API 26 (Android 8.0—Oreo).¹ The autofill service was intended to provide OS-level support for password manager autofill, obviating the need to rely on the accessibility service. With the autofill service, the OS manages communication between autofill services, such as password managers, and autofill clients, which are the apps that need to be filled. By default, the autofill service relies on autofill clients to annotate their login interfaces using the `android:autofillHints` attribute, though the autofill service does have several backup heuristics it can use to identify login forms if the application is not properly annotated.

On Android, the password managers are free to style the autofill credential selection dialog as they see fit. Figure 4.3 gives two examples of different UIs on Android.

OpenYOLo Framework

Around the same time the autofill service was released, Dashlane (a password manager) and Google worked together to create the OpenYOLo framework.² Similar to the autofill service, OpenYOLo was designed to address problems with using the accessibility service to implement autofill. In OpenYOLo, rather than modifying the Android framework itself (as was done for the autofill service), clients (apps) and servers (password managers) are modified so that the app can receive credentials directly from the password manager. The advantage of OpenYOLo over the autofill service is that OpenYOLo is deterministic and allows an app to specify which details it wants to retrieve from a credential provider. However, because OpenYOLo requires more effort to implement than and was not designed to be interoperable with the autofill service, it has not seen wide adoption.³

¹25% of Android devices are unable to use the autofill service as they have not, and likely cannot, be upgraded to API 26.

²<https://github.com/openid/OpenYOLo-Android>

³<https://discussions.agilebits.com/discussion/111985/is-openyolo-dead>

B Password Manager Download Statistics

In this section, we present the figures detailing app download statistics for the password managers we studied on iOS and Android. On Android we used the download count from the Google Play Store (see Figure 1). Because iOS does not provide detailed information about downloads from the App Store, we used estimates for April 2020 from SensorTower (see Figure 2).⁴

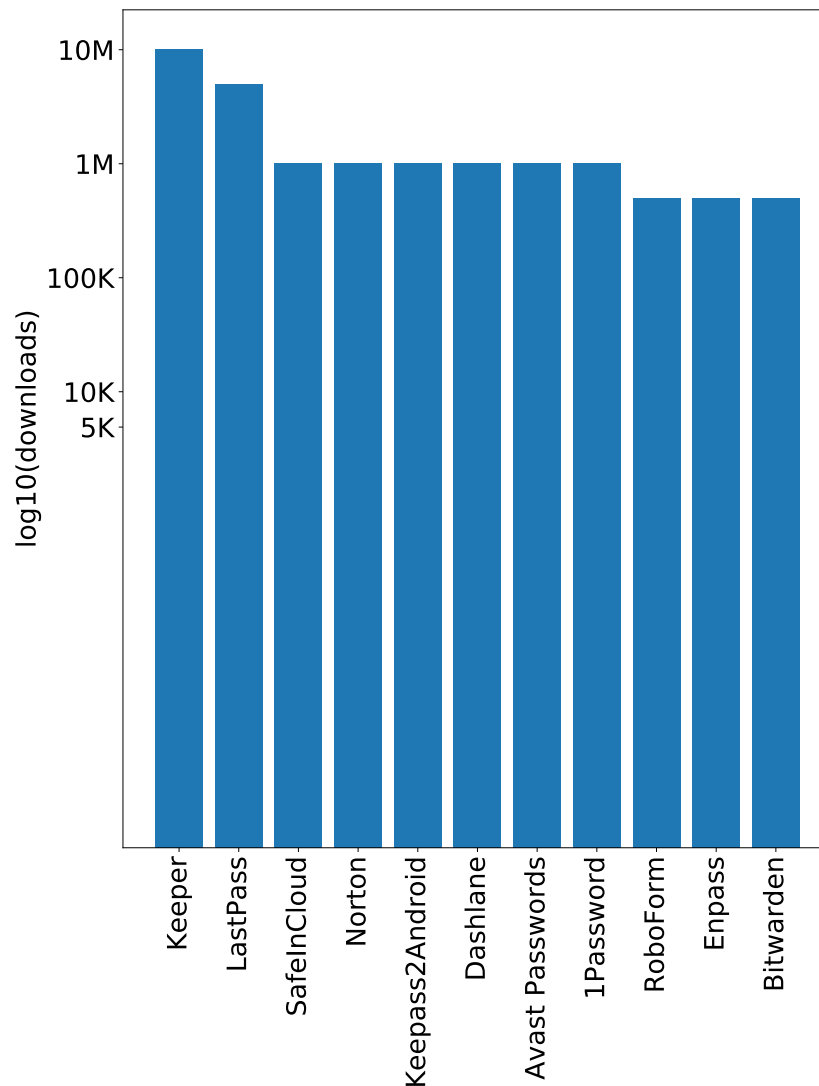


Figure 1: Google Play Store Downloads from March 2020

⁴<https://sensortower.com/>

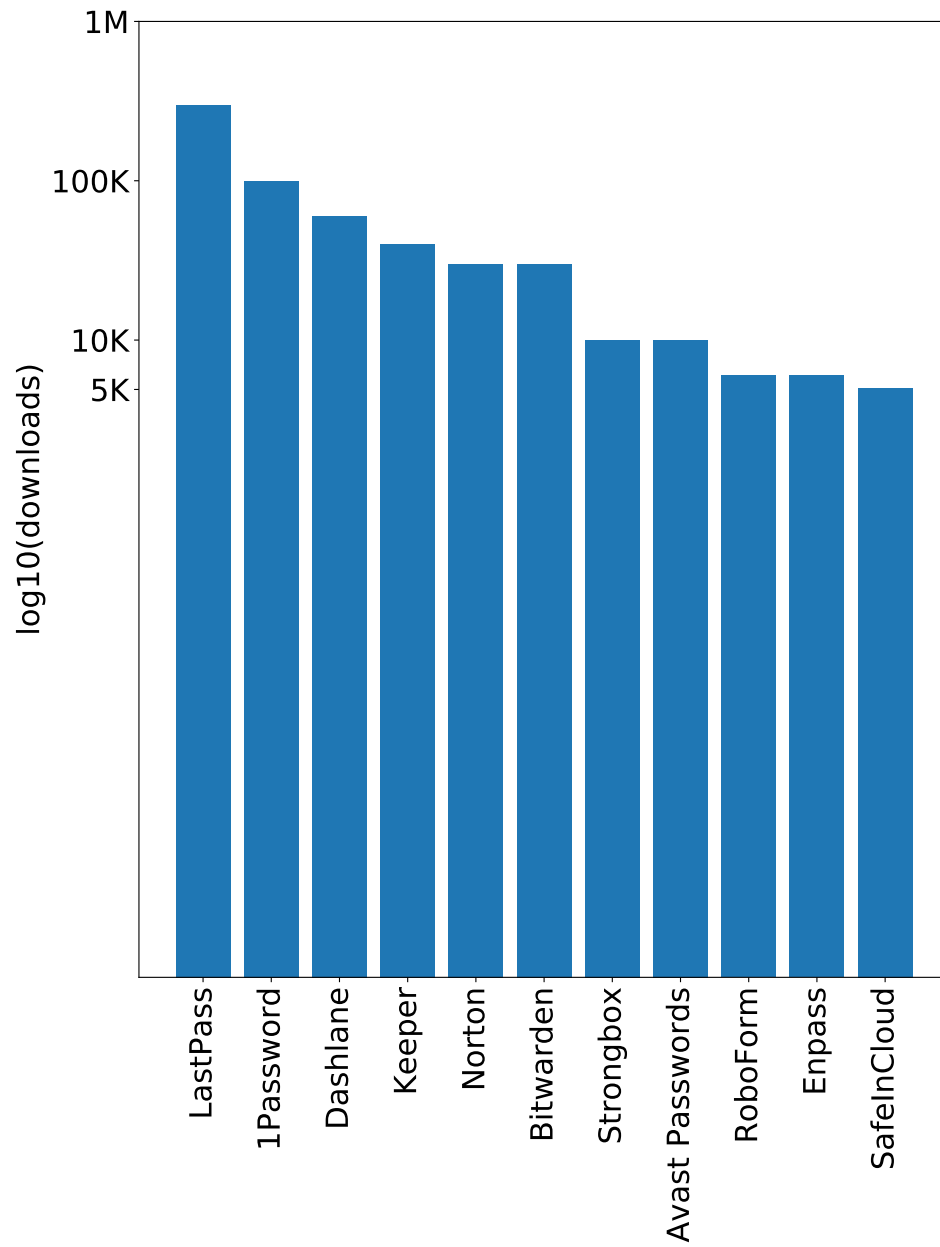


Figure 2: iOS Download Estimates from April 2020

C Password Manager Browser Evaluation Results

To test browser autofill for the mobile frameworks, we evaluated the security of fourteen different managers implemented with those frameworks. In this section, we given detailed results for each manager using the evaluation criterion identified by Oesch and Ruoti [48]. Table 1 gives the results of this evaluation for iOS and Table 2 gives the results for Android. Note that in Table 1 there is only a single row for iOS autofill, that is because that framework completely handles the autofill experience for managers, and thus there are no results for individual managers. In contrast, both iOS app extensions and the Android autofill service allow the managers to have some limited control over the autofill process.

In addition to testing the various mobile managers, we also tested the password managers integrated into several mobile browsers. While these managers do not use the system-wide autofill frameworks, they act as a point of comparison for the managers implemented with the mobile frameworks. These browser managers only work within their respective browser, and do not support generic app-based autofill.

Table 1: Autofill in mobile browsers on iOS

System		Interaction required for HTTPS	Interaction required for bad cert	Interaction required for HTTP	Won't fill same-origin iframe	Won't fill cross-origin iframe	Won't fill different URL path	Won't fill HTTPS→bad cert	Won't fill HTTPS→HTTP	Won't fill different action (static)	Won't fill different action (dynamic)	Won't fill different method	Won't autofill different input fields	Won't fill type="text" field	Won't fill non-login form fields	Won't fill invisible password field	Fills password on transmission	Obeys autocomplete="off"
Password AutoFill		●	●	●	○	○	○	○	○	○	○	○	○	●	○	○	○	○
App extensions	1Password	●	●	●	●	●	○	○	●	○	○	○	○	●	○	○	○	○
	Avast	●	●	●	●	●	○	○	○	○	○	○	●	●	○	○	○	○
	Bitwarden	●	●	●	●	●	○	○	○	○	○	○	○	●	○	○	○	○
	Enpass	●	●	●	○	●	○	○	○	○	○	○	○	○	○	○	○	○
	Keeper	●	●	●	○	●	○	○	○	○	○	○	○	●	○	○	○	○
	LastPass	●	●	●	●	●	○	○	○	○	○	○	○	●	○	○	○	○
	Norton	●	●	●	●	●	○	○	○	○	○	○	○	●	●	●	○	○
	RoboForm	●	●	●	●	●	○	○	○	○	○	○	○	●	○	○	○	○
Browser	Chrome	○	●	●	○	●	○	○	●	○	○	○	○	●	○	○	○	○
	Firefox	○	○	●	●	●	○	○	●	○	○	○	○	●	○	○	○	●
	Edge	●	●	●	●	●	○	○	○	○	○	○	○	●	○	○	○	○

● Secure behavior ● Partially secure behavior ○ Insecure behavior

Table 2: Autofill in mobile browsers on Android

System		Interaction			iframe		Difference in fill form								Fields			Misc	
Autofill service	1Password	●	●	●	○	◐	○	○	○	○	○	○	○	○	○	○	○	○	○
	Avast Passwords	●	●	●	○	◐	○	○	○	○	○	○	○	○	○	●	○	○	○
	Bitwarden	●	●	●	○	◐	○	○	○	○	○	○	○	○	○	○	○	○	○
	Dashlane	●	●	●	○	◐	○	○	○	○	○	○	○	○	○	●	○	○	○
	Enpass	●	●	●	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○
	Keeper	●	●	●	○	◐	○	○	○	○	○	○	○	○	○	○	○	○	○
	LastPass	●	●	●	○	◐	○	○	○	○	○	○	○	○	○	○	●	○	○
	Norton	●	●	●	○	◐	○	○	○	○	○	○	○	○	○	●	○	○	○
	RoboForm	●	●	●	○	◐	○	○	○	○	○	○	○	○	○	●	○	○	○
	SafeInCloud	●	●	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	Smart Lock	○	○	○	○	○	○	●	●	○	○	○	○	○	○	●	○	○	○
Browser	Chrome	○	○	○	○	○	○	●	●	○	○	○	○	○	○	●	○	○	○
	Firefox	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○
	Opera	○	○	○	○	○	○	●	●	○	○	○	○	○	○	●	○	○	○

● Secure behavior ● Partially secure behavior ○ Insecure behavior

D Android Credential Mapping Details

In this section, we give additional details on how password managers handled mapping apps and the domain's associated with passwords stored in the password manager.

1Password, **Enpass**, **Keepass2Android**, and **RoboForm** require users to manually associate apps and domains. Of these three, only RoboForm warns users of the danger that manual association can cause.

Avast maintains a SQLite database with two relevant tables. The `domain_info` table contains a list of 1,203 websites whose package names are a simple inversion of their website address. For example, `facebook.com` is matched with `com.facebook`. If the first two components of an app's package name are in this table, then the app is considered to match (e.g., `com.walmart.evil` is matched to `walmart.com`). The second table, `alternate_mapping` is a static mapping for apps that do not use a simple name inversion. For example, this table maps `ign.com` to `com.mobile.ign`.

Bitwarden uses a simple heuristic that looks for substring matches between the domain and the components of the app's package name, though it does ignore components that are TLDs (e.g., `.com`, `.org`). For example, `com.wal.evil` would match domains that contained `wal` or `evil`—for example, `walmart.com`.

Dashlane maintains a list of 285 mobile apps, their associated domains, and a SHA-512 hash of their signing certificates. If an app is not on this list, Dashlane will not even offer to autofill it. Unique to Dashlane, mapping behavior changes if the user turns off the autofill service and only enables the accessibility service. In this case, instead of checking the list of allowed apps, Dashlane uses a simple heuristic that compares the components of the package name to domains looking for matches (ignoring common components such as `com` and `android`). If a match is found, a warning is shown to the user informing them that they are autofilling an “unverified app”.

SafeInCloud uses a simple heuristic that considers the first two components of the package name and matches those against domains. For example, `com.walmart.evil` will match `walmart.com`. While this does require malicious apps to use the same prefix as a legitimate app, we have confirmed that it is possible to upload such apps to the PlayStore.

Smart Lock maps apps by downloading the DAL files for the stored domains. It did not allow a user to build an association between a website and an application.

Keeper uses a more complex heuristic to establish its credential mappings. First, it will query the app store for information about the application. If the app is found, Keeper will use the **app developer website** field as the domain for the app. If the app is not found, the user will need to manually associate the app to a domain. As first reported by Aonzo et al [3], this heuristic is vulnerable to attackers who lie about the app developer website, something which is not verified when apps are uploaded. We verified this by uploading an app to the Play store with the developer website set to **walmart.com** and checking that Keeper does indeed offer to fill Walmart's credentials into our app. We note that Keeper does show a warning in this case.

Lastpass contains a SQLite database that maps apps and domains. Additionally, like Lockwise and SafeInCloud it uses a simple heuristic that considers the first two components of the package name and matches those against domains. Unlike those two, if no match is found the user is then prompted to pick which domain should be matched with the app. If the user does so, they are then asked if they want to share this mapping with other users. If enough users share this mapping, it will be auto suggested by LastPass to other users in the future (crowdsourced mappings). LastPass does warn users when they first associate an app and a domain.

Norton includes a static file (**resources/assets/theirpartyapp.properties**) mapping 131 package names to domains. If an app is not on this list, Norton will not show an autofill dialog, not even to inform the user about the lack of a match.

E Observational Study Instruments

E.1 Screening Survey

Q1. Which password manager do you use?

◦ *LastPass* ◦ *Bitwarden* ◦ *KeePass* ◦ *1Password* ◦ *Dashlane* ◦ *Other*

Q2. On what devices do you use your password manager? Select all that apply.

◦ *Desktop PC* ◦ *Laptop* ◦ *Phone* ◦ *Tablet* ◦ *Other*

Q3. Would you be willing to participate in a one-on-one Zoom interview with our research group about how you use your password manager? You will receive compensation for your time. The interview will be less than one hour and will be recorded. As part of the interview, you will share your screen and demonstrate how you use your password manager to complete several tasks. Your interview data will be anonymized and kept confidential.

◦ *Yes* ◦ *No*

Q4. Please rate the following statements about your password creation and usage. Options: Never, Rarely, Sometimes, Often, Always

◦ *I do not change my passwords, unless I have to.* ◦ *I use different passwords for different accounts that I have.* ◦ *When I create a new online account, I try to use a password that goes beyond the site's minimum requirements.* ◦ *I do not include special characters in my password if it's not required.*

Q5. On a 5-point scale ranging from 1 = not at all concerned to 5 = very concerned, how concerned are you about. . .

◦ *How concerned are you about institutions, public agencies, or intelligence services monitoring your online communication?* ◦ *How concerned are you about website or app providers sharing your data with unknown third parties?* ◦ *How concerned are you about other people getting information about you without your consent?* ◦ *How concerned are you about someone misusing your identity on the Internet?* ◦ *How concerned are you about other people spreading information about you without your knowledge?*

Q6. Do people ask you for computer-related advice?

- *Yes* ◦ *No*

Q7. Do you know at least one programming or scripting language?

- *Yes* ◦ *No*

Q8. How often do you use the programming or scripting languages you know?

- *Daily* ◦ *Weekly* ◦ *Monthly* ◦ *Rarely*

Q9. What is your gender?

- *Male* ◦ *Female* ◦ *I prefer not to answer* ◦ *Other*

Q10. What is your age?

- *Under 21* ◦ *21-34* ◦ *35-44* ◦ *45-54* ◦ *55-64* ◦ *Above 64* ◦ *I prefer not to answer*

Q11. What is the highest level of school you have completed or the highest degree you have received?

- *Less than a high school degree* ◦ *High school or GED* ◦ *Some college*
- *Trade/vocational/technical* ◦ *Associates (2-year)* ◦ *Bachelors (4-year)* ◦ *Masters*
- *Professional* ◦ *Doctorate* ◦ *I prefer not to answer*

Q12. Please specify your ethnicity.

- *Hispanic or Latino* ◦ *American Indian or Alaska Native* ◦ *Asian* ◦ *Black or African American*
- *Native Hawaiian or Other Pacific Islander* ◦ *Caucasian or White* ◦ *Multiracial*
- *Other* ◦ *Prefer not to say*

E.2 Interview Sign Up Survey

Q1. Rate the following statements regarding device security. Options: Never, Rarely, Sometimes, Often, Always

- *I set my computer screen to automatically lock if I don't use it for a prolonged period of time.*
- *I use a password/passcode to unlock my laptop or tablet.*
- *I manually lock my computer screen when I step away from it.*
- *I use a PIN or passcode to unlock my mobile phone.*

Q2. Rate the following statements regarding general security habits. Options: Never, Rarely, Sometimes, Often, Always

- *When someone sends me a link, I open it without first verifying where it goes.*
- *I know what website I'm visiting based on its look and feel, rather than by looking at the URL bar.*
- *I submit information to websites without first verifying that it will be sent securely (e.g., SSL, "https://", a lock icon).*
- *When browsing websites, I mouseover links to see where they go, before clicking them.*
- *If I discover a security problem, I continue what I was doing because I assume someone else will fix it.*

Q3. Rate the following statements regarding software updates. Options: Never, Rarely, Sometimes, Often, Always

- *When I'm prompted about a software update, I install it right away.*
- *I try to make sure that the programs I use are up-to-date.*
- *I verify that my anti-virus software has been regularly updating itself.*

Q4. Enter your MTurk ID below so that we can compensate you \$26 after the interview.

Q5. Signup for an interview using this link: <https://calendly.com/userlabutk/pwminerview>

Q6. Enter the meeting ID that you received for your Zoom meeting in the box below.

Q7. Do you use a password manager on your mobile device?

- *Yes*
- *No*

If you use your password manager on your mobile device, watch one of the following instructional videos on how to share your screen in Zoom from a mobile device. Please make sure to install Zoom on your mobile device and know how to share your screen prior to the interview if you use your password manager on your mobile device.

Below are some resources that can help you get setup:

1. Instructions Share Screen Android - 0:12-0:50 - <https://www.youtube.com/watch?v=H0uHVjJQ740>
2. [InstructionsShareScreeniOS-0:21-0:37,1:02-1:13-https://www.youtube.com/watch?v=aydCQcYtpwc&t=90s](https://www.youtube.com/watch?v=aydCQcYtpwc&t=90s)
3. <https://support.zoom.us/hc/en-us/articles/115005890803-iOS-Screen-Sharing>

E.3 Semi-structured Interview Script

Opening

“First off, thanks for participating in our research. As a reminder, we are going to be recording this session. Is that still Ok?” (If no) Let them know that unfortunately they will be unable to participate in the study. You may now end the study without paying them.

“My name is [redacted] and I am a research assistant at the University of Tennessee. In my research group we are trying to understand how people use technology to secure their lives. Our goal is to create security tools that better meet the needs of users like yourself. In this study, we are trying to understand how you use your password manager, [NAME], so that we can improve it to better fit your needs. So the main goal of this study is to understand the way that you use [NAME] in your life.

There are no right answers or responses. We really just want to see how you normally use [NAME]. So just do whatever you would normally do. And if something is hard for you, that is a problem with [NAME] and not with you. Our goal is to make [NAME] better, so any problems you have using it will help us know the ways that [NAME] can be improved. And

if at any point in the interview something pops into your mind that you think would help us understand how you use [NAME], please feel free to stop me and share your thoughts.

“Now, let’s talk about the structure of the interview just so you know what to expect. The entire interview should take under an hour. I’m going to start by asking you some basic questions about how you use [NAME].

After that, we’re going to ask you to share your screen and show us how you use [NAME] on your own device. To protect your privacy and ensure we don’t accidentally see any account information, before you share your screen in the interview, we’re going to have you log out of [NAME]. We’ll then have you login to [NAME] using an email and password that we provide to you.

“Next, I’ll ask you to create an account on several websites the same way you normally would in real life, except that you will use an email address I provide to protect your privacy. The reason for doing that is that we want to understand how you use [NAME] when you create and manage accounts. We’re not going to ask you to do anything other than create, log in to, or log out of an account on any of the websites we visit. And we’ll be deleting all of these accounts after the interview.”

Then, if you use your password manager on your mobile device, I’ll have you join this Zoom call with your phone and you will complete a few tasks on your phone with [NAME]. At that point we’ll close with a few final questions and then wrap up.”

If you need any help sharing your screen with Zoom or have any additional questions, please let me know. We just want you to focus on helping us understand how you use [NAME], so just let me know if you run into any issues.

“Before we begin the interview, do you have any questions for me?

“Great! Let’s get started.”

Pre-interview Checklist:

1. Enable multiple participant screen sharing
2. Ask participant to disable webcam
3. Ask participant to log out of accounts on all relevant devices

4. Verify participant is able to share their screen from all relevant devices
5. Start Recording

Interview

Generic questions about usage and login

“First, let’s discuss how you use your password manager on your desktop or laptop.”

1. “How often do you use [NAME]?”
2. “How did you pick [NAME] as your password manager?”
3. “How did you pick your master password?”
4. “Do you use anything in addition to your master password to protect your [NAME] account?”
5. Do you ever suggest that other people use [NAME]? Do you promote password managers to friends or family?” If yes - “What does that look like? What is your motivation for doing so?” If no - “How come?”
 - (a) Do you know anyone who has adopted as a result?
6. Do you ever save your passwords anywhere other than [NAME], such as in Chrome or Firefox?
 - (a) If yes - “Why do you use both? How do those two work together? Do you have the same passwords in both?”

Account setup

1. “When was the last time you set up [NAME] on a new device?”
2. Before we continue, you’ll need to log out of [NAME]. If you need any guidance on how to do that just let me know.

3. Could you please share your screen now. Wait for them. Could you now login to your password manager using the login information I just sent you over chat. Wait for them.”
 - (a) “Assuming you had just installed this password manager, would you change any of the default settings before using it? Could you show us which ones?”

Register a new account

1. “We’re now going to have you set up an account on Reddit. Have you ever used Reddit before?”
 - (a) If no - “Alright, so Reddit is a social media platform where users share news and custom content that they’ve created.”
 - (b) If yes - “If you are currently logged in to reddit.com, we’ll have you log out before creating a new account.”
2. “Now, go ahead and navigate to reddit.com and sign up for a new account with the email I provided. After we are done with the interview, we will delete this Reddit account.”
 - (a) Now that you’re done, go ahead and log out of Reddit.
3. “Now we’re going to have you perform a similar task for ebay.com.”
 - (a) Now that you’re done, go ahead and log out of ebay.com.
4. “Do you ever experience problems when trying to save new accounts in [NAME]?”
5. “Do you normally store credentials for sites like Reddit and Ebay in your password manager? More generally, what types of accounts do you store in [NAME]?”
 - (a) How do you store passwords if you don’t save them in [NAME]?
6. Thank them

Logging in and updating an existing account

1. “Now let’s go back to Reddit and login”
2. “Do you ever experience problems when trying to login to websites that you’ve saved in [NAME]?”
3. Now let’s say you need to update your password for Reddit. Show us how you would go about updating your password for reddit.”
 - (a) If they seem to be searching for a password reset in settings, tell them how to get there.
4. Now do the same for Ebay
5. How often do you update passwords?

Other common tasks

1. Creating a password
 - (a) “When you want to create a new password, how do you normally go about doing that?”
 - (b) “Do you create passwords differently for different types of websites?”
2. Autolock and log out
 - (a) How often do you log out of your password manager? (desktop/phone)
 - (b) Does your password manager ever automatically log you out/lock? What is your opinion of that feature?
3. App Autofill
 - (a) Do you ever use your pwm to fill apps on your desktop?
 - (b) (If yes) Do you ever encounter issues when doing so?
4. Sharing a password

- (a) “If you wanted to share a password with someone, how would you normally go about doing that?”
- (b) “Did you know that you can share passwords directly with other [NAME] users?”
- (c) If no - “Would you like to check it out?”
- (d) After showing - “Is this a feature you’d like to use in the future?”

5. Password health check

- (a) “If you wanted to check if any of your passwords were weak or had been compromised in a data breach, how would you normally go about that?”
- (b) “Did you know that [NAME] can help you identify weak and compromised passwords?”
- (c) If no - “Would you like to check it out?”
- (d) After showing - “Is this a feature you’d like to use in the future?”

6. “We’ve gone through using [NAME] to create and use accounts, as well as some additional features. Are there any other features of [NAME] you use that we haven’t covered? Can you show us?”

“You can stop screen sharing”

Mobile usage

1. “Do you use [NAME] on your mobile phone?”
 - (a) If no, skip rest of section
2. “What are some of the differences between how you use [NAME] on your desktop versus on your phone?”
 - (a) “Do you create accounts for websites on your mobile device?” If so, “Do you ever encounter problems?”

- (b) “Do you use your password manager to log in to apps on your phone?” If so, “Do you ever counter issues when using [NAME] to log into apps?”
 - (c) “Do you generate passwords on your mobile device?” If so, “How does your experience of generation on mobile devices compare to that on the desktop?”
3. If they actually use any functionality on their phone - ”We want to see how it works for you on mobile. We’re going to have you join this Zoom call on your phone and share your screen.”
 4. Could you please login to Reddit now.
 - (a) Great, thanks. Could you log out of reddit now. This is the last we’ll use reddit.
 5. If they create accounts on their phone - “We’re going to have you create an account again. This time it’s going to be for memrise.com, a language learning platform.
 - (a) “Alright, go ahead and create an account for memrise.com”
 - (b) If they autofill apps on their phone - “Great, now let’s try using the credentials you just created to log in to the memrise app. Please download the memrise app on your device and login using the credentials that you just saved in your password manager.”

You can stop sharing your screen.

Other remarks

1. “Thanks for participating in our interview. Before we wrap up, is there anything left you would like to share that you think would be helpful for us to understand?”
2. “If there was one feature you could add or thing you would change about [NAME], what would it be?”
3. If anything else you think would be helpful occurs to you at a later time, feel free to send us an email at userlab@utk.edu.

Explain compensation

1. We will pay you via a bonus on mturk
2. Let's verify mturk id for payment - what is your full mturk id? You can just paste that into the chat if that's easier for you.

E.4 Consent Form for Interview

Overview

Research study title: Password manager usage interview Time commitment: Signing up for an interview will take about 2 minutes; the interview will take between 30–60 minutes. Compensation: You will be paid \$1 for signing up for the interview. After completing the interview, you will be paid \$25 as a bonus to this hit. Requirements: You must be age 18 or older to participate in this study.

In our research group, we are trying to understand how people use password managers so that we can make them more usable and secure. As part of this effort, we are conducting interviews with individuals regarding their password manager usage. Based on your answers to an earlier screening survey, you have been selected to participate in these interviews.

What will I do in this study?

In this study you will conduct an interview regarding your password manager usage. This interview will take place over Zoom. During the interview, we will ask that you share your screen at several points so that you can demonstrate how you use your password manager. We will take precautions to avoid recording any sensitive information during the screen shares. You are not required to share video from your webcam during the interview.

Can I say "No"

Being in this study is up to you. After completing the interview, it will not be possible to remove your responses as we will delete any data linking your responses to you.

Are there any risks to me?

We don't know of any risks to you from being in the study.

Are there any benefits to me?

We do not expect you to directly benefit from being in this study. Your participation may help us to learn more about password managers and how they are used, providing indirect benefits.

What will happen with the information collected for this study?

As part of this study, we collect your name and email address so that we can communicate with you. We also keep records of audio from your interview and screen recordings of you using your password manager on your own devices. This We will keep and publish results from this study, including quotes from participants' interviews.

The information collected in this study will be analyzed to identify how to improve the usability and security of password managers. We will record the audio from your interview and the video from the screen recordings, but no other video data. This data is only accessible to research staff at the USER Lab and the raw data will be destroyed at the completion of our research.

These results will be published and possibly presented at scientific meetings. We will clean this data to ensure that this data contains no personally-identifiable information. When referring to participant statements, we will do so using an anonymized label (e.g., 'R12 said, ...').

Will I be paid for being in this research study?

You will be paid \$1 for signing up for the interview. After completing the interview, you will be paid \$25 as a bonus to this hit.

Who can answer my questions about this research study?

If you have questions or concerns about this study, or have experienced a research related problem or injury, contact the researcher, [redacted] at [redacted].

For questions or concerns about your rights or to speak with someone other than the research team about the study, please contact:

Institutional Review Board The University of Tennessee, Knoxville 1534 White Avenue
Blount Hall, Room 408 Knoxville, TN 37996-1529 Phone: 865-974-7697 Email:
utkirb@utk.edu

Statement of Consent

I have read this form, been given the chance to ask questions and have my questions answered. If I have more questions, I have been told who to contact. By clicking the link to signup for an interview, I am agreeing to be in this study. I can print or save a copy of this consent information for future reference. If I do not want to be in this study, I can close my internet browser.

Interview signup: [link](#)

E.5 Consent Form for Screening Survey

Overview

Research study title: Password manager screening survey for interviews Time commitment: This survey will take about 5–10 minutes to complete. Compensation: You will be paid \$1 for your participation. Requirements: You must be age 18 or older to participate in this study.

In our research group, we are trying to understand how people use password managers so that we can make them more usable and secure. As part of this effort, we want to conduct interviews with individuals regarding their password manager usage. This survey is intended to help us identify potential participants for these interviews.

What will I do in this study?

You will answer demographic questions and questions about your usage of password managers.

Can I say "No"

Being in this study is up to you. After completing the study, we cannot remove your responses because we will delete any information linking your response to your turker ID after we have selected our interview participants.

Are there any risks to me?

We don't know of any risks to you from being in the study.

Are there any benefits to me?

We do not expect you to directly benefit from being in this study. Your participation may help us to learn more about password managers and how they are used, providing indirect benefits.

What will happen with the information collected for this study?

The information from this survey will be used to identify potential participants for an interview on their usage of their password manager. If you are selected, we will offer you a second HIT that can be used to signup and participate in the interview.

Overall demographics and responses to the survey questions will be published and possibly presented at scientific meetings. These results will be anonymous and no one will be able to link your responses back to you. As such, please do not include your name or other information that could be used to identify you in your survey responses.

Will I be paid for being in this research study?

You will be paid \$1 when you complete the survey.

Who can answer my questions about this research study? If you have questions or concerns about this study, or have experienced a research related problem or injury, contact the researcher, [redacted] at [redacted].

For questions or concerns about your rights or to speak with someone other than the research team about the study, please contact:

Institutional Review Board The University of Tennessee, Knoxville 1534 White Avenue
Blount Hall, Room 408 Knoxville, TN 37996-1529 Phone: 865-974-7697 Email:
utkirb@utk.edu

Statement of Consent

I have read this form, been given the chance to ask questions and have my questions answered.
If I have more questions, I have been told who to contact. By clicking the survey link below,
I am agreeing to be in this study. I can print or save a copy of this consent information for
future reference. If I do not want to be in this study, I can close my internet browser.

Survey Link - [survey link](#)

Survey code Provide the code given to you when completing the survey:

Vita

Sean Oesch is a security researcher in the areas of usable security, systems security, and network security. His work at the University of Tennessee aims to make authentication more secure by offering concrete solutions to problems with password manager design and implementation. At UTK, he also explored the use of nation scale mobile ad hoc networks to enable communication over large areas when it would otherwise be impossible, such as after natural disasters or in heavily censored regions. His work at Oak Ridge National Laboratory as a research scientist spans everything from using images of vehicles at stoplights to enhance fuel efficiency in cities via machine learning based optimization algorithms to the design of experiments to test the usability and efficacy of emerging cyber technologies in a state of the art cyber range.