That Was Then, This Is Now: A Security Evaluation of Password Generation, Storage, and Autofill in Browser-Based Password Managers

Sean Oesch and Scott Ruoti, *University of Tennessee*

https://www.usenix.org/conference/usenixsecurity20/presentation/oesch

This paper is included in the Proceedings of the 29th USENIX Security Symposium.

August 12–14, 2020

978-1-939133-17-5

Open access to the Proceedings of the 29th USENIX Security Symposium is sponsored by USENIX.

That Was Then, This Is Now: A Security Evaluation of Password Generation, Storage, and Autofill in Browser-Based Password Managers

Sean Oesch
University of Tennessee, Knoxville
toesch1@vols.utk.edu

Scott Ruoti
University of Tennessee, Knoxville
ruoti@utk.edu

Abstract

Password managers have the potential to help users more effectively manage their passwords and address many of the concerns surrounding password-based authentication. However, prior research has identified significant vulnerabilities in existing password managers; especially in browser-based password managers, which are the focus of this paper. Since that time, five years has passed, leaving it unclear whether password managers remain vulnerable or whether they have addressed known security concerns. To answer this question, we evaluate thirteen popular password managers and consider all three stages of the password manager lifecycle—password generation, storage, and autofill. Our evaluation is the first analysis of password generation in password managers, finding several non-random character distributions and identifying instances where generated passwords were vulnerable to online and offline guessing attacks. For password storage and autofill, we replicate past evaluations, demonstrating that while password managers have improved in the half-decade since those prior evaluations, there are still significant issues; these problems include unencrypted metadata, insecure defaults, and vulnerabilities to clickjacking attacks. Based on our results, we identify password managers to avoid, provide recommendations on how to improve existing password managers, and identify areas of future research.

1 Introduction

Despite the well-established problems facing password-based authentication, it continues to be the dominant form of authentication used on the web [4]. 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 [12, 26]. In fact, with the increase in the number of passwords users are required to store, they often reuse passwords across websites [11, 15, 25, 33]. 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 [18]. However, the number of data breaches is on the rise [28], and this situation leaves many users vulnerable to exploitation.

Password managers can help users more effectively manage their passwords. They reduce the cognitive burden placed upon the user by generating strong passwords, storing those passwords, and then filling in the appropriate password when a site is visited. The user is now able to follow the latest security advice regarding passwords without placing a high cognitive burden on themselves. But password managers are not impervious to attack. Li et al. [19] previously found significant vulnerabilities in major password managers like LastPass and RoboForm. Both Silver et al. [29] and Stock and Johns [31] demonstrated that browser-based password managers, including LastPass and 1Password, are vulnerable to cross-site scripting attacks (XSS) and network injection attacks as a result of their password autofill features.

Since these studies five or more years have passed, 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. 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.

In our evaluation, we consider the full password manager lifecycle [8]—password generation (Section 4), storage (Section 5), and autofill (Section 6). 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 [17] and autofill [19, 29, 31].

Our results find that while password managers have improved in the past five years, there are still significant security concerns. We conclude the paper 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 7).

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 paper 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 [17]) and is the only study that simultaneously considers all three stages of the password manager lifecycle [8]—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 [17, 19, 29, 31] 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.

2 Background

In this section, we describe the responsibilities of a password manager. We 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 [19]. 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.

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.

2.2 Related Work

Several studies have looked at various aspects of password manager security.

Web Security Li et al. [19] analyzed the security of five extension-based password managers, finding significant vulnerabilities in the tools as well as the websites that hosted the user's password vault. These vulnerabilities included logic and authorization errors, misunderstandings about the web security model, and CSRF/XSS attacks. They also found that password managers that were deployed using

bookmarklets did not use iframes properly, leaving the tools vulnerable to malicious websites.

Google's Project Zero found a bug in LastPass where credentials from the last visited site could be leaked to the currently visited site; this bug has since been fixed.[1]

Autofill. Silver et al. [29] 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 [31] also studied autofill related vulnerabilities in six browser-based password managers and had similar findings to Silver et al.

Storage. Gasti and Rasmussen [17] 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.

Chatterjee et al. [6] and Bojinov et al. [2] proposed alternative password vault schemes that are more resilient to offline attacks, but password managers have not adopted these schemes.

A recent study by Independent Security Evaluators [13] 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.

Usability. In 2006, Chiasson et al. [7] conducted a usability study of two password managers, finding significant vulnerabilities due to users' incomplete mental models regarding how these password managers worked. More recently, Fagan et al. [14] surveyed users and non-users of password managers to better understand why people chose to adopt password managers. They found that users adopted password managers primarily due to usability, not security benefits; in contrast, non-users generally avoid password managers due to security, not usability concerns.

Lyastani et al. [20] studied whether adoption of a password manager helped increase the strength of a user's passwords, finding that while users of password managers on average had stronger passwords than those of the general public, they still rarely had a unique, brute force-resistant password for every website. Zhang et al. [36] interviewed users to investigate how they use their password managers, finding that users of browser-based managers were more likely to reuse password than users of app-based or extensions-based password managers.

Relation to This Work To our knowledge, our 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 [8] (i.e., generation, storage, and autofill). Much of the work examining the security of password manager autofill and storage is now over five or more years old [17, 19, 29, 31]. As there have been significant updates to password managers in that time, we have replicated this early work to determine whether the password managers we studied have addressed the core issues revealed by this prior work, or whether they remain vulnerable.

3 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 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 [5]), 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 App

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

KeePassX (v2.0.3). KeePass is an app-based password manager originally built using the .NET platform and

[1] https://bugs.chromium.org/p/project-zero/issues/detail?id=1930

	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	◐	●		●	○	●	○	●	○		◐

Table 1: Analyzed Password Managers

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 [13].

3.2 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 [13]. 1Password provides both an app-based client (1Password) and an extension-based client (1Password X); in this paper, 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.

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 [13]. 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 [13], 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 paper, we evaluated the extension-based client for the same reason we took this approach with 1Password X.

3.3 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.

[2] https://support.1password.com/getting-started-1password-x/

[3] https://www.roboform.com/business/features

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.

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

3.4 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.

4 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 Generator[4] (SPG), the first search result when searching for "password generator" on Google.

4.1 Settings and Features

Table 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 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 paper.

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.

4.2 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, KeeyPassXC), 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

[4] https://passwordsgenerator.net

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	chrm	>1	●	○	15	all		!-.:_
Safari	sfri	15	●	○	15	all		-
SPG	psgn	6–2048	●	●	16	all		!"#$%&'()*+,-./:;<=>?@[\]^_`{}~|
/dev/random	dvrn	>1	○	○				!"#$%&'()*+,-./:;<=>?@[\]^_`{}~|␣

Table 2: Overview of Password Generation Features

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 zxcbvn password analysis tool [34], and a recurrent neural net-based password guesser [22].

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 [3], 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.

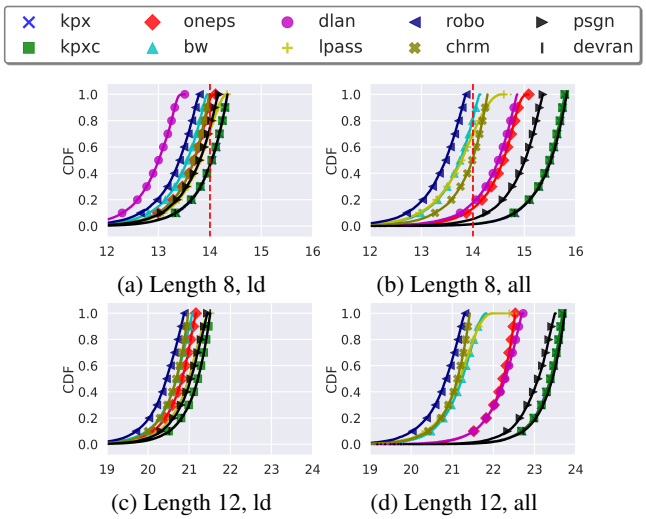
The χ^2 test for randomness is a simple statistical test for determining whether the difference between two distributions can be explained by random chance. We used the χ^2 test to evaluate each of our passwords sets independently and corrected our p-values using a Bonferonni correction[5] to account for the multiple statistical tests from the same family.

The zxcbvn tool created by Daniel Wheeler [34] is used to detect dictionary words and simple patterns that might be present in passwords, both potential examples of non-randomness. zxcbvn 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. [22]. 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 1 in Appendix A. 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 zxbcvn 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

[5] To 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.



(a) Length 8, ld (b) Length 8, all

(c) Length 12, ld (d) Length 12, all

Figure 1: Neural Network Guess Estimates (log_{10}). Differences are primarily attributed to character set size.

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 [16]. Using this guess count, we were able to analyze whether the password managers were generating passwords that were vulnerable to these attacks.

4.3 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 1c and 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 1a and 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 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), 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 4, detailed χ^2 and p values are in Tables 2–9 in Appendix A). 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 1). When this feature is enabled, the probability is also proportional to the number of characters in that character set (see Equation 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. We note that it would be possible to adjust for this skew and preserve a uniform distribution, though there no significant security effect from not correcting it.

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

$$((length - |sets|) * \frac{1}{|characters_{all}|}) + \frac{1}{|characters_{set}|} \tag{2}$$

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, 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 (see Table 10 in Appendix A) 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 zxcbvn results, we found that occasionally all password managers would generate exceptionally weak passwords, examples of which are shown in Table 5. 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

System	Characters Sorted by Frequency
kpx	`'+,7lFr[AE/8"$OdNzGMn`_*3;D:i|Z@s=#]whRb6~&Wm(2ck)\g^oy<aL}JCTq4e!->VI1BPvY9HSUjp{?5%xt0fX.uQK`
kpxc	`NtpgT@vO<Be1hiY)H-`Kk;IXu^c4z$yqo6F/r>S_%Z3+U[=DL\as"0(2'VA?PdRm.:*jb]W~}Exn{f!Q|7#CJw8G,&91M5`
oneps	`0314569782>^:*@.-~+%?,_V=a}N]!d)YjZK#ubeCATJUGBEDyozrgkMRtHwLvXWmqxfQhsniPFpc`
bw	`%7#9532^@46!$&8*IYBomtbJFLUPVnXdzSexagHZrwusiMkpqcWNRvQKhfDCGAjyTE`
dlan	`5473698QRHDNPAFMBKSCLYTXEGJijepnfgtryhbdkmqsxacz_*@~=){'[;&,!#.:"/$(^|+}-%]?o`><\wuU2WvVZ`
dlan*	`3498576QNBHPAFMXJCYKTGSDRLEdqzmnpsfjghbtxckaioyre/$#{!<-,?"(\=].~*^'+`|:;}>_)[%@&`
lpass	`%!$#&*@^jAGfRMOYPobszleTUiIwVhtDKNQqJgBSaWmpudcnLkEyHrZFxCXv3987542160`
robo	`%#@!^$8624375HLYJXPDFCWAUENKVSTiRQGBMydgstkvqpfnjbwaemhrucox9Zz`
chrm	`umSHDMeYNbnEGzCwaspZg6f:!XqLTBWrR9t5h3JP8Q7jc_iAFVK-kdxv2Uy.4`
psgn	`4239750618LoQPYliRHpJkqIUZOnWBxmNhvdDbgAXtuVcwzysSCarMjEGKTfeF\!/(.+%}@|'=[$`{?:*>&)~-;"^],<#_`
dvrn	`.\zdAP4L0^W,6@&+3w%?ebSqc-"Y$8EM'~QVu}iGojv(tK:y;I>#<TD_aU9C[lrH)/h5Z1␣|sR`=mO]{*xXgnBNpfFJk2!7`

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

Table 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	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 4: χ^2 test for random character distribution

System	Length	Composition	Guesses (log_{10})	Password
KeePassX	8	l	4.96	`TaKEdeen`
KeePassXC	8	sd	4.84	`'+'+'+_+`
1Password X	12	ls	8.76	`oMMMMMMT?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`

Table 5: Randomly Generated Weak Passwords

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.

5 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 6.

5.1 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

System	Storage	Storage Encryption				Metadata Encrypted									
		Encryption	KDF	KDF Rounds	Requires strong MP	URL	Icon	Username	Creation time	Modification time	Last use time	Fill count	User's email	User's 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					●		●	●	●	●	●	●	●	

[1] On Linux, Chromium-based browser attempt to store the password in the GNOME keyring or KWallet 4. If neither of these are available, it will store the passwords in plaintext [9].

Table 6: Overview of Password Vault Encryption

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).

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 paper, we also note that there is a need for more research examining the security of OS-provided encryption functions and keychains.

5.2 Metadata Privacy

Compared to earlier findings by Gasti and Rasmussen [17], 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.

6 Password Autofill

Of the password managers we evaluated, only KeePassX did not support autofill in the browser[6] 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. [19], Silver et al. [29], and Stock and Johns [31]. We also updated these attacks to address protections that have been added by browsers and password managers since the attacks were first described. Table 7 highlights several of our findings.

6.1 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 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).

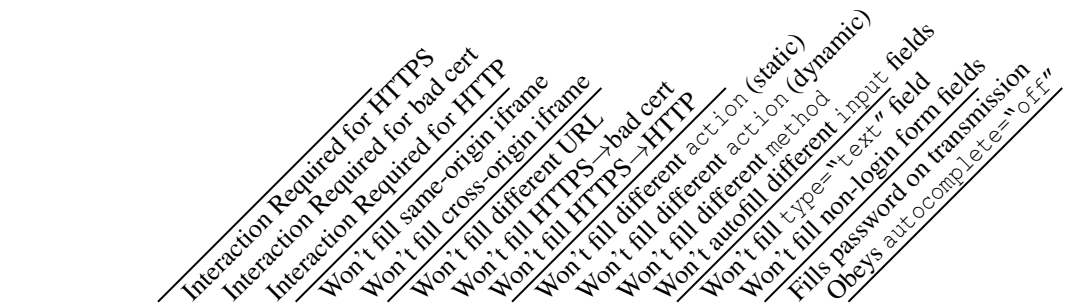
While network injection attacks are still possible on sites using HTTPS (i.e., TLS man-in-the-middle attacks [24]), they are much easier to accomplish and more likely if the HTTPS certificate is invalid. Reasons for a bad HTTPS certificate range from benign (e.g., expired by a day) to malicious (e.g., 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.

6.2 Autofill for iframes

Autofilling passwords within iframes is especially dangerous, regardless of whether user interaction is required or not [29, 31]. 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

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



System	Interaction			iframe		Difference in fill form							Fields		Misc	
KeePassXC	◐	◐	◐	◐	○	○	○	●	●	●	○	○	●	●	○	○
1Password X	●	●	●	◐	●	◐	◐	●	◐	◐	◐	◐	●	●	○	○
Bitwarden	◐	◐	◐	◐	●	○	○	○	○	○	○	○	●	○	○	○
Dashlane	○	○	●	◐	●	○	○	●	○	○	○	○	○	●	○	○
LastPass	○	○	●	◐	●	○	○	◐	●	○	○	◐	●	●	○	○
RoboForm	◐	◐	◐	◐	●	○	○	○	○	○	○	○	●	●	○	○
Chrome	○	●	○	○	◐	○	●	●	○	○	○	○	●	○	○	○
Edge	○	●	●	○	◐	○	◐	●	○	○	○	○	●	○	○	○
Firefox	○	○	○	○	○	○	○	●	●	●	○	○	●	○	○	○
IE	○	●	●	○	◐	○	◐	●	○	○	○	○	●	○	○	○
Opera	○	●	○	○	◐	○	●	●	○	○	○	○	●	○	○	○
Safari	●	●	●	◐	◐	◐	◐	◐	◐	◐	◐	◐	◐	●	●	○	○

Table 7: Overview of Password Autofill Features

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 [29, 31]

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.

6.3 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.

6.4 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.

6.5 Potential Mitigation

Stock et al. [31] 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 7.

Silver et al. [29] and Stock and Johns [31] 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 [32]. 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. [23]. This helps explains why no password managers currently obey this parameter, even though in prior research there was some support for this attribute in browsers [29, 31].

6.6 Web Vault Security & Bookmarklets

In their analysis of extension-based password managers, Li et al. [19] 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

[7] It 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.

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 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 [19]. 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. [19].

7 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 [17, 19, 29, 31]. 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 [29, 31]. 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 [29, 31] (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.

7.1 Recommendations

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.

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.

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.

7.2 Future Work

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 [1, 35]), and preventing malicious websites from creating look-alike phishing interfaces [27].

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 [31] (see Section 6.5); (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 [30].

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.

HTML-Supported Password Generation. Stajano et al. [30] recommended adding HTML attributes to help password managers identify the policy to use when generating passwords. We believe that this approach should receive more attention. In particular, it would be helpful to see developer studies studying the feasibility adding this feature to existing websites and user studies to ensure that this feature is understandable and helpful to users. It would also be worth examining whether such annotations could be automatically inferred and added by semantically evaluating the code that checks passwords.

Mobile Password Managers. Our work examined the security of password managers in a desktop environment. Given the prevalence of mobile devices, a similar analysis of the security of mobile password managers is necessary.

8 Conclusion

Password managers are currently being recommended by the media [10, 21]; as such, it is disappointing that users need to be cautious when selecting a password manager and must

also spend time to ensure that they understand how to correctly configure it. As experience has shown, pushing these responsibilities onto users rarely has the expected outcome [18]. Therefore, we believe it is important that researchers continue to evaluate the progress of password managers—both in terms of security and usability—and that work is done to continue to improve the security and usability of password managers [27].

Disclosure

We have made these results available to the maintainers of each password manager studied. RoboForm has already adopted several of our recommendations.

Research Artifacts

The generated data, scripts used to analyze that data, and all analysis artifacts are available for download at https://userlab.utk.edu/papers/oesch2020that.

Acknowledgments

The authors would like the thank their shepherd Ben Stock and the anonymous reviewers for their helpful feedback.

References

[1] S.M. Bellovin and M. Merritt. 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, 1992.

[2] Hristo Bojinov, Elie Bursztein, Xavier Boyen, and Dan Boneh. Kamouflage: Loss-resistant password management. In *European symposium on research in computer security*, pages 286–302. Springer, 2010.

[3] Joseph Bonneau. The science of guessing: analyzing an anonymized corpus of 70 million passwords. In *2012 IEEE Symposium on Security and Privacy*, pages 538–552. IEEE, 2012.

[4] Joseph Bonneau, Cormac Herley, Paul C Van Oorschot, and Frank Stajano. 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, 2012.

[5] Cristian Bravo-Lillo, Lorrie Cranor, Julie Downs, Saranga Komanduri, Stuart Schechter, and Manya Sleeper. Operating system framed in case of mistaken identity: measuring the success of web-based spoofing attacks on os password-entry dialogs. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 365–377. ACM, 2012.

[6] Rahul Chatterjee, Joseph Bonneau, Ari Juels, and Thomas Ristenpart. Cracking-resistant password vaults using natural language encoders. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 481–498. IEEE, 2015.

[7] Sonia Chiasson, Paul C van Oorschot, and Robert Biddle. A usability study and critique of two password managers. In *USENIX Security Symposium*, volume 15, pages 1–16, 2006.

[8] Yee-Yin Choong. 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, 2014.

[9] Chromium. Linux password storage. https://chromium.googlesource.com/chromium/src/+/master/docs/linux_password_storage.md, 2019. Accessed: 2019-05-20.

[10] CNET. The best password managers of 2019. https://www.cnet.com/news/the-best-password-managers-directory/. Accessed: 2019-02-22.

[11] Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov, and XiaoFeng Wang. The tangled web of password reuse. In *NDSS*, volume 14, pages 23–26, 2014.

[12] Matteo Dell'Amico, Pietro Michiardi, and Yves Roudier. Password strength: An empirical analysis. In *2010 Proceedings IEEE INFOCOM*, pages 1–9. IEEE, 2010.

[13] Independent Security Evaluators. Password managers: Under the hood of secrets management. https://www.securityevaluators.com/casestudies/password-manager-hacking/, 2019. Accessed: 2019-02-22.

[14] Michael Fagan, Yusuf Albayram, Mohammad Maifi Hasan Khan, and Ross Buck. An investigation into users' considerations towards using password managers. *Human-centric Computing and Information Sciences*, 7(1):12, 2017.

[15] Dinei Florencio and Cormac Herley. A large-scale study of web password habits. In *Proceedings of the 16th international conference on World Wide Web*, pages 657–666. ACM, 2007.

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

[17] Paolo Gasti and Kasper B Rasmussen. On the security of password manager database formats. In *European Symposium on Research in Computer Security*, pages 770–787. Springer, 2012.

[18] Cormac Herley. 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, 2009.

[19] Zhiwei Li, Warren He, Devdatta Akhawe, and Dawn Song. The emperor's new password manager: Security analysis of web-based password managers. In *USENIX Security Symposium*, pages 465–479, 2014.

[20] Sanam Ghorbani Lyastani, Michael Schilling, Sascha Fahl, Michael Backes, and Sven Bugiel. Better managed than memorized? studying the impact of managers on password strength and reuse. In *27th USENIX Security Symposium*, pages 203–220, 2018.

[21] PC Magazine. The best password managers of 2019. https://www.pcmag.com/roundup/300318/the-best-password-managers. Accessed: 2019-02-22.

[22] William Melicher, Blase Ur, Sean M Segreti, Saranga Komanduri, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Fast, lean, and accurate: Modeling password guessability using neural networks. In *25th USENIX Security Symposium*, pages 175–191, 2016.

[23] Mozilla. The autocomplete attribute and login fields. https://developer.mozilla.org/en-US/docs/Web/Security/Securing_your_site/Turning_off_form_autocompletion#The_autocomplete_attribute_and_login_fields, 2019. Accessed: 2019-11-12.

[24] Mark O'Neill, Scott Ruoti, Kent Seamons, and Daniel Zappala. Tls proxies: Friend or foe? In *Proceedings of the 2016 Internet Measurement Conference*, pages 551–557. ACM, 2016.

[25] Sarah Pearman, Jeremy Thomas, Pardis Emami Naeini, Hana Habib, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, Serge Egelman, and Alain Forget. Let's go in for a closer look: Observing passwords in their natural habitat. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 295–310. ACM, 2017.

[26] Shannon Riley. Password security: What users know and what they actually do. *Usability News*, 8(1):2833–2836, 2006.

[27] Scott Ruoti and Kent Seamons. End-to-end passwords. In *Proceedings of the 2017 New Security Paradigms Workshop*, pages 107–121. ACM, 2017.

[28] Security Scorecard. Statistics: Cybersecurity data breaches on the rise. https://securityscorecard.com/blog/cybersecurity-data-breaches-statistics-on-the-rise, 2018. Accessed: 2019-02-22.

[29] David Silver, Suman Jana, Dan Boneh, Eric Yawei Chen, and Collin Jackson. Password managers: Attacks and defenses. In *USENIX Security Symposium*, pages 449–464, 2014.

[30] Frank Stajano, Max Spencer, Graeme Jenkinson, and Quentin Stafford-Fraser. Password-manager friendly (pmf): Semantic annotations to improve the effectiveness of password managers. In *International Conference on Passwords*, pages 61–73. Springer, 2014.

[31] Ben Stock and Martin Johns. Protecting users against xss-based password manager abuse. In *Proceedings of the 9th ACM symposium on Information, computer and communications security*, pages 183–194. ACM, 2014.

[32] W3C. Html. https://www.w3.org/TR/html52/sec-forms.html#element-attrdef-autocompleteelements-autocomplete, 2019. Accessed: 2019-11-09.

[33] Ke Coby Wang and Michael K Reiter. How to end password reuse on the web. *arXiv preprint arXiv:1805.00566*, 2018.

[34] Daniel Lowe Wheeler. zxcvbn: Low-budget password strength estimation. In *25th USENIX Security Symposium*, pages 157–173, 2016.

[35] T. Wu et al. The secure remote password protocol. In *Internet Society Symposium on Network and Distributed System Security*, 1998.

[36] Shikun Aerin Zhang, Sarah Pearman, Lujo Bauer, and Nicolas Christin. Why people (don't) use password managers effectively. In *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*, 2019.

A Additional Password Generation Data

	all		l		ld		ls		sd	
System	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

Figure 2: Length 8 χ^2 Scores for Character Frequency

	all		l		ld		ls		sd	
System	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 8: Length 12 χ^2 Scores for Character Frequency

	all		l		ld		ls		sd	
System	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

Table 9: Length 20 χ^2 Scores for Character Frequency

Length	Composition	Characters Sorted by Frequency
8	all	%!^@$#4627583NPHFDJUXACTSGMERBKLQVYWqmgasneokfvptbuhyixdwrcjzZ9
12	all	$%#^!@2637548BHGFSQECXWYJRDNMUALVPTKdtboenhskjvqaicgwpmxfyur9zZ
20	all	%#@!^$8462735XHVPJWCUFKLYNDESAMTQRiBGgdveaspnkytqjfbmxwrcuoh9Zz
8	l	GHDYEQKPJFURCTASNLVMXBWpyikuvmtofxecasdwjngbhqrZ
12	l	VMDFQAGNRLUEXKCJSBPWTYHcmfiqyawnektsdvrgjhopxbuZ
20	l	REFQWJUTBKDGCMAHSVPXYLNfkvyjsnhwoepabqixgdturcmZ
8	ld	5782346RUALJDQFHSPKEVGTMYBXCNWhynabrqwpkfumxjvctodsigeZ9
12	ld	6853247JUWYSBLTQFGCRMPVKANXHEDgcidbjtwpesafxqvhmrkounyZ9
20	ld	6532874MTJFSVCYDNHPLGWEXQABUnRkeKswpjughytdqbircafovxm9Z
8	ls	%@^$#!SLFWVAURKNTEXDQJYBMHPGCavhtndwcjkyufxieqobrgpmszZ
12	ls	$^%#@!FHJVESBGMUYXDLTPCAQNWRKrwogjhicexmsyftvkqdabupnZz
20	ls	%@$^#!PFAXTKBQCSHDGVJEMWRYtNgUfLabyshrkpwmdouvqxjineczZ
8	sd	#$@%!^65324879
12	sd	$@!#^%57263489
20	sd	@^!$#%63582749

Table 10: Character Frequencies of Generated Passwords from RoboForm

```
{
 "args": {
   "pwd_file": ["$TRAINING_FILE"],
   "pwd_format": ["list"],
   "log_file": "$LOG_FILE",
   "arch_file": "$ARCH_FILE",
   "weight_file": "$WEIGHT_FILE"
 },

 "config": {
   "intermediate_fname":
        "$INTERMEDIATE_FILE",
   "min_len": $PASSWORD_LENGTH,
   "max_len": $PASSWORD_LENGTH,

   "training_chunk": 1024,
   "layers": 2,
   "hidden_size": 1000,
   "dense_layers": 1,
   "dense_hidden_size": 512,
   "generations": 5
 }
}
```

```
{
 "args": {
   "enumerate_ofile": "$GUESSES_FILE",
   "log_file": "$LOG_FILE",
   "arch_file": "$ARCH_FILE",
   "weight_file": "$WEIGHT_FILE"
 },

 "config": {
   "guess_serialization_method":
        "delamico_random_walk",
   "password_test_fname": "$TESTING_FILE",
   "parallel_guessing": true,

   "intermediate_fname": "$INTERMEDIATE_FILE",
   "min_len": $PASSWORD_LENGTH,
   "max_len": $PASSWORD_LENGTH,

   "training_chunk": 1024,
   "layers": 2,
   "hidden_size": 1000,
   "dense_layers": 1,
   "dense_hidden_size": 512,
   "generations": 5
 }
}
```

Listing 1: Neural Network Configuration—Training (Left) and Testing (Right)