Leaky Autofill: An Empirical Study on the Privacy Threat of Password Managers' Autofill Functionality

Yanduo Fu, Ding Wang
College of Cyber Science, Nankai University, Tianjin 300350, China; wangding@nankai.edu.cn
Key Laboratory of Data and Intelligent System Security (NKU), Ministry of Education, Tianjin 300350, China
Tianjin Key Laboratory of Network and Data Security Technology, Nankai University, Tianjin 300350, China

2024 Annual Computer Security Applications Conference (ACSAC) | 979-8-3315-2088-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/ACSAC63791.2024.00037

***Abstract*—Password managers (PMs) provide users with convenient and robust functionalities to manage their credentials, highly recommended by security experts and major standard bodies. One of the most popular features is the autofill functionality, with which users need *a single click or a few clicks* to fill in every field in web forms, facilitating the process of completing web forms. However, such *indiscriminate autofill* brings severe privacy threats. PMs may inadvertently fill data into wrong fields in web forms, even *hidden* fields, potentially leading to privacy leaks and credential theft.**

In this paper, we conduct an empirical study evaluating the effectiveness of 30 popular PMs in identifying and handling hidden `<input>` fields. We focus on the privacy threats posed by the autofill functionality, which fills data into hidden fields. We develop a *semi-automated* autofill testing tool and explore whether PMs autofill sensitive data into hidden fields across 15 concealment techniques and three web forms, including personal information, credit card, and login forms. Experimental results reveal that *every* PM autofills data into hidden fields in at least one web form, with an overall filled probability of 58.7% in 1032 scenarios. Further analysis reveals that login forms are the most vulnerable, with a 65.7% probability of hidden fields autofill. Hidden fields concealed by `clip-path` and `content-visibility` are filled with passwords by all PMs. Besides, built-in-browser PMs exhibit a 4.07 times higher likelihood of filling data into hidden fields than separately-installed PMs. Even more concerning, built-in-browser PMs, except Safari, autofill passwords into hidden fields under *any* concealment technique. 37.7% of autofill scenarios with insufficient user interaction pose heightened privacy threats, as users are *unaware* of autofill content. These privacy threats have been confirmed by popular PMs like LastPass.

To mitigate the threats brought by the autofill functionality, we present two actionable recommendations for PM operators/developers: (1) providing fine-grained data types in rendered overlays before autofilling; (2) integrating visual language model techniques to accurately identify fillable fields and prevent data autofilling into hidden fields. We believe this work makes a substantial step toward understanding the security implications of the autofill functionality in PMs.

***Index Terms*—User Authentication, Password Manager, Autofill**

1. Introduction

Passwords play a pivotal role in human's digital life. The number of password accounts that common users maintain has increased to 80-112 per user before COVID-19 [18], [24], [43]. During the pandemic, with more and more services made online and people working at home, NordPass reports that this number has increased to 168 in 2024 [60]. Such an increase poses a significant cognitive burden, and exacerbates users' vulnerable behaviors of choosing popular passwords [61], reusing passwords [42], [63], and utilizing personal information to create passwords [62]. All this poses significant security threats [2], [29], [66].

Password managers (PMs), serving as a solution to mitigate the security and usability dilemma of passwords, have gained increasing adoption due to their ability to store and manage login credentials securely, alleviating the burden of remembering and managing multiple credentials (e.g., username and password) [58]. A desirable feature driving PM usage is its convenient *autofill functionality* [30], [45], [48], [49], capable of filling various web forms automatically. Users typically need *a single click or a few clicks* to trigger the autofill functionality. Then, the PM fills in *every detected field* in the web form using stored data, significantly reducing the effort of repetitive manual input [65]. Most PMs provide login credential autofill functionality. Advanced PMs (e.g., 1Password [32] and LastPass [25]) also support autofilling personal information, credit card details, passports, driver licenses, and other information, being a well received input method for users [28].

While convenient, recent studies indicate that such *indiscriminate autofill* can be exploited by attackers, leading to privacy leaks and credential theft. At PETS'20, Acar et al. [4] summarized their previous findings in 2017 [3] and reported that malicious third-party scripts can insert invisible login forms into web pages. Built-in-browser PMs may imprudently autofill stored usernames and passwords into injected forms on page load. Then, the credential is sent to the attacker-hosted remote server. At ACM CCS'20, Lin et al. [27] showed that built-in-browser PMs and two popular separately-installed PMs (i.e., 1Password [32] and LastPass [25]) fail to identify *hidden* fields (e.g., covered by overlays) when filling in personal information and credit card details in web forms. This leads to the leakage of

2576-9103/24/$31.00 ©2024 IEEE
DOI 10.1109/ACSAC63791.2024.00037

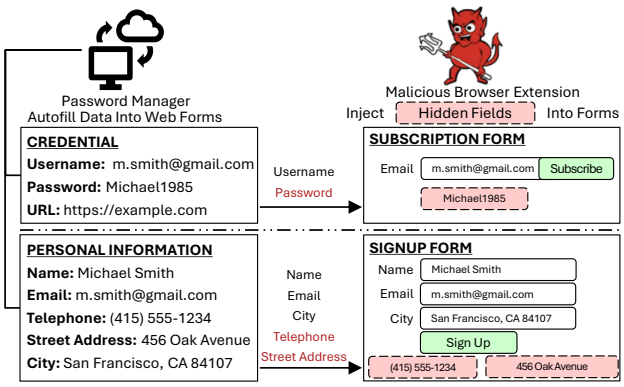


Figure 1: Our threat model of malicious browser extensions: the attacker injects hidden fields into forms, misleading the PM into erroneously detecting the form type. This exploitation of the autofill functionality allows the extensions to obtain sensitive data that should not have been filled in.

sensitive information (e.g., phone, address, and credit card number) while users are only aware of *visible* fields for less-sensitive information (e.g., username and credit card name). Looking further back, at AsiaCCS'14, Stock and Johns [57] demonstrated that XSS attackers could drive PMs to autofill credentials in login forms and obtain filled passwords. They also discussed real-world threats by exploring the autofill functionality of six built-in-browser PMs.

Motivations and Research Questions. Prior research [4], [27], [57] has revealed that built-in-browser PMs struggle to detect hidden form fields accurately and can be exploited by client-side attackers (e.g., XSS attackers), leading to user privacy leaks and credential theft. With the increasing adoption of separately-installed PMs (e.g., 1Password [32] and LastPass [25]) [58], a natural question arises: *Do these perceived more secure [30], [48] separately-installed password managers possess a more robust ability to detect and handle hidden form fields?* In this paper, we conduct the *first* empirical research on the autofill functionality of separately-installed PMs, examining their ability to detect and handle hidden fields and measuring the extent of real-world security and privacy threats raised.

An attack example. To gain an intuitive grasp of such threats, we consider an attacker capable of injecting hidden fields into web pages to collect sensitive information, such as malicious browser extensions. Taking 1Password browser extension version 2.23.3 [32] as an example, the PM displays a PM icon in the email field of the subscription form and renders an overlay similar to that of the login form (see Figure 6(c) in Appendix B). The attacker can inject a hidden password field in the subscription form and the PM would then autofill the stored password into the injected field without modifying the web page layout or alerting users. When sensitive information is filled into unexpected hidden fields, the credentials are likely obtained by malicious extensions.[1] Figure 1 presents the attack method and operations.

1. We have reported this privacy threat to 1Password. They respond that this practice involves performance and user experience consideration.

This attack can stealthily exfiltrate sensitive information without users' awareness, especially when the autofill functionality lacks sufficient user interactions (e.g., presenting the types of data to be autofilled) and cannot effectively detect hidden fields. This also violates the "transparency" security principle in GDPR [15] and the "right to know" privacy rights in CCPA [9] when handling user data.

Thus, it is critical to evaluate separately-installed PMs' ability to detect and handle hidden fields and the strength of their interaction with users. In all, our work aims to address three main research questions (RQ):

RQ1: Do password managers provide detailed information about the ready-to-filling data and require user interaction when providing the autofill functionality?

RQ2: How well do separately-installed password managers detect hidden fields concealed by various techniques? Do results differ among web form types?

RQ3: Do separately-installed password managers behave better than built-in-browser password managers? Have built-in-browser password managers improved since the vulnerability disclosure at ACM CCS'20 [27]?

Methods. To answer the above questions, we first select 30 popular PMs (including 24 separately-installed PMs) and explore their autofill functionality. Specifically, we focus on the autofill functionality triggering method and user interaction strength. The latter includes the detailed information displayed in the rendered overlay and whether the PM presents warnings or requires re-authentication before autofilling (RQ1). Then, we expand the eight concealment techniques of Lin et al. [27] to fifteen and develop a semi-automated tool to assess whether PMs autofill stored sensitive information into invisible `<input>` fields hidden using 15 concealment techniques in three web forms, including personal information, credit card, and login forms.

Next, we conduct a χ^2 test on our experimental results to understand the differences in the filled probability between different PM types, form types, and concealment techniques (RQ2). We employ linear regression and decision tree analysis to gain further insights. We also assess whether there have been any improvements in built-in-browser PMs since 2020 [27] (RQ3). Besides, we analyze whether there is a difference when adding hidden style to `<input>` fields' ancestors instead of fields themselves. Finally, we study the visibility check mechanism of five open-source PMs.

Findings. The tested 30 PMs provide the autofill functionality in 69 out of 90 scenarios (=3 form types $\times$ 30 PMs). All PMs in credit card forms exhibit strong interaction with users. However, in 26 scenarios, the interaction strength with users is insufficient: PMs in 53.3% (=16/30) of login forms exhibit weak interaction, either autofilling on page load (10) or rendering overlays without any details about the filled data (9), with three failing in both scenarios. For personal information forms, one PM does not inform users of the form type, and nine PMs only inform users of the form types without specifying the filled data types.

Moreover, 83.3% (=25/30) PMs can detect hidden fields concealed via `display:none` and `visibility:none`

CSS properties, and HTML `hidden` property in at least one form. However, the other 12 (=15-3) techniques are challenging to detect, such as fields covered by overlay and with `clip-path` property. Separately-installed PMs are 4.07 (=(216/51)/(390/375), see Sec. 4.1 for details) times less likely to autofill data into hidden fields than built-in-browser PMs. Login forms are the most vulnerable due to weak user interaction and poor detection of hidden fields. Moreover, there are no significant improvements in the detection mechanism of built-in-browser PMs compared with Lin et al.'s findings [27]. There is also no notable difference between hiding `<input>` fields by directly adding properties to the fields and their ancestor nodes. These findings underscore the serious privacy threats and credential theft risks posed to users when weak user interactions accompany the autofill functionality into hidden fields.

We have reported our findings to several PM operators. LastPass [25] has confirmed our findings but has not resolved them. Other operators, such as 1Password [32] and Bitwarden [7] who have responded to us, claim that comprehensive visibility checking may compromise performance and user experience.

Contributions. We make the following contributions:

(1) *New findings and insights.* We conduct an empirical study on the autofill functionality of 24 popular *separately-installed* PMs, *for the first time* focusing on whether they autofill data into web forms containing hidden fields. We also expand Lin et al.'s work [27] on six built-in-browser PMs. Our research reveals that no PM can completely avoid autofilling data into hidden fields across all three web forms and 15 concealment techniques. Login forms and personal information forms with insufficient user interaction are more likely to be exploited by attackers. PM operators could learn from our new findings to balance the security and usability of their visibility-checking mechanisms.

(2) *A semi-automated tool for password manager (PM) autofill functionality testing.* We develop the *first*, to our knowledge, semi-automated PM autofill functionality testing tool, which significantly alleviates the testers' burden of manually clicking and recording data. Although our tool aims to detect whether PMs autofill data into hidden fields, common operations, including triggering the autofill functionality and recording the filled results, are quite similar. Thus, our tool can be applied in recent empirical PM studies [20], [21], [41].

(3) *New perspective of countermeasures.* We propose two actionable recommendations to PM operators to mitigate threats brought by the autofill functionality: (1) providing fine-grained details of filled data type in the PM-rendered overlay and (2) employing the visual language model technique to analyze which web form fields should be filled with stored data.

Ethical Considerations. All experiments are conducted within a controlled environment designed for our research purposes, and the tested accounts belong to testers. Thus, our experiments cause no harm to other PM users and real-world website users. All personal information shown in all figures is fake and generated randomly.

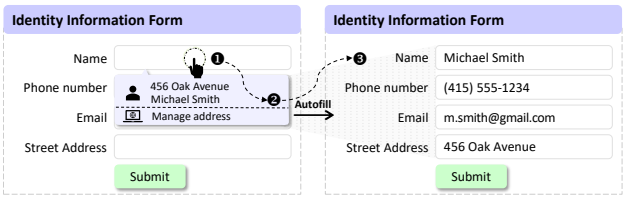


Figure 2: Users need to click the form field to trigger the PM's autofill functionality. Then, users could click the rendered overlay to fill the data into the form automatically.

Open Source. We release the source code of our semi-automated autofill testing tool at https://github.com/Leaky-Autofill/LeakyAutofill-Artifact with detailed documents.

2. Background and Threat Model

In this section, we first introduce the background on password managers and the autofill functionality, and then present the threat model and scope considered in this paper.

2.1. Password Managers

Password managers (PMs) assist users in handling password management tasks, such as securely memorizing passwords using encrypted storage, generating strong and unique passwords, and facilitating password input through autofill functionality. These tools are highly recommended by security experts [22], [31] and major standard bodies [8], [12], [40], and have gained a higher adoption rate among common users recently [58]. In this paper, we focus on the autofill functionality in web forms and categorize PMs into two categories: (1) Built-in-browser PMs (e.g., Chrome, Safari[2], and Firefox) are integrated into web browsers and function as a browser module, gaining higher adoption rate among users [58]; (2) Separately-installed PMs (e.g., 1Password [32] and LastPass [25]) are external applications that require installation and generally provide password management services via browser extensions.

2.2. Autofill Functionality

Most PMs provide autofill functionality to assist users in filling information into web forms, such as credentials, personal information, credit card details, and passports. To use the autofill functionality, users first need to store the information in the PM by manually entering and importing it, or through prompts for storage upon initial input on web forms. When revisiting pertinent web pages, PMs render an overlay, and prompt users to autofill their information. Generally, users need a single click (or a few clicks) to fill every field in a web form. For instance, Figure 2 presents a personal information web form. When (1) the user clicks on the `name` field and triggers the autofill functionality, (2) she

2. Prior studies [30], [37], [45], [48] consider Safari's PM as part of OS-integrated PMs, provided by the macOS's KeyChain. However, as we focus on web scenarios, we regard Safari as a browser-based PM here.

only needs one click on the rendered overlay, and (3) the PM automatically fills stored information into every detected field in the web form. Note that credentials are bound to the domain names. Thus, the password stored for website A cannot be autofilled into website B. In contrast, personal information and credit card details are associated with the PM account and can be autofilled across various websites when the information is stored in the PM.

During autofilling, PMs identify the form type and fields to be filled using various methods (e.g., Fathom for Firefox [35]), then autofill each field matched with stored data. In this paper, we consider that PMs can correctly detect form and field types, and focus on the privacy threats posed by the autofill functionality in identifying hidden fields in forms (i.e., PMs may mistakenly fill sensitive data into hidden fields, leading to user privacy leaks and credential theft).

In this paper, we select 24 popular separately-installed PMs based on the active user count. Additionally, we include the six browsers in Lin et al.'s study [27] to evaluate the differences between separately-installed PMs and built-in-browser PMs. We also explore whether there are improvements in the latter.

2.3. Threat model and scope

We consider an attack scenario involving hidden HTML fields on web pages. It consists of two cases.

First, curiosity-driven websites aim to acquire sensitive information without users' awareness by injecting concealed fields into the webpage and exploiting the autofill functionality to obtain the filled sensitive information. In this case, curiosity-driven websites are *less* likely to collect users' credentials but other sensitive information such as address, telephone, and credit card numbers. This aligns with the threat model of Lin et al.'s [27]. Their study shows that when considering personal information forms and credit card forms in the Chrome browser, 3.3% (2,776 out of 83,054) of web pages have hidden fields filled by browsers. Moreover, Senol et al. [51] have shown that web trackers on thousands of websites collect users' email addresses before form submission. This indicates that curiosity-driven websites could leverage the scenario that PMs autofill data into hidden fields to obtain users' sensitive information.

Additionally, we consider malicious extensions or any third parties capable of injecting hidden fields into web forms. For instance, in Figure 1, users have stored personal information and credentials in PMs. By employing concealment techniques (e.g., `clip-path:inset(100%);` `position:absolute` CSS property), adversaries can inject concealed `<input>` fields into targeted forms on page load without modifying the webpage layout. Then, when users trigger the autofill functionality, the corresponding data will be filled into the injected fields. Our observation reveals that most (61 out of 69) autofill scenarios fill data into `<input>` fields hidden using the `clip-path` property. Consequently, these hidden fields are erroneously filled with users' sensitive data, enabling malicious extensions to obtain user-sensitive information. In this case, attackers are not limited to stealing credentials on login pages but also obtaining sensitive information filled in registration pages.

In real-world scenarios, such malicious extension requires host-permissions with `<all_urls>` to conduct the attack on as many websites as possible or requests several URLs for targeted attacks (in a version 3 `manifest.json` file), but requires no further sensitive permissions. According to the study at AsiaCCS'24 [19], the host-based permission `<all_urls>` is the most popular among benign Chrome browser extensions and the third most popular among vulnerable extensions. Besides, a recent study at WWW'24 [39] highlights that malicious extensions can bypass the Chrome Web Store's security check by masking extensions with benign purposes. Several malicious extensions have been found to be active within the Chrome Web Store in recent years [59]. Thus, our considered malicious extensions are feasible and practical in real-world scenarios.

The essential exploitation of this attack is that PMs autofill data into hidden fields. Note that although browser extensions are popular and easily distributed through the Web Browser Store and there are some studies [34], [39] to conduct the attack through malicious third-party scripts, we do not focus on how the attacker tricks users into running third-party scripts or installing malicious extensions on the client side. In contrast, we aim to reveal real-world security and privacy threats posed by the autofill functionality. Thus, we explore the strength of user interaction in the autofill functionality, including the autofill triggering method and details displayed in overlays. When PMs have weak interaction with users (i.e., PMs do not inform users which data will be filled or autofill on page load), sensitive data will be leaked without user awareness. Besides, as the PM's adoption rate is increasing [58] and autofill is one of the most favorable features [30], [45], [48], the threat brought by the autofill functionality will arise real-world risks.

3. Methodology

We first present the password manager (PM) selection process. Then, we conduct experiments to evaluate the default autofill functionality of selected PMs across three types of web forms: login forms, personal information forms, and credit card forms. Finally, we assess whether PMs would inadvertently autofill data into hidden fields concealed using various techniques.

3.1. Choosing Password Managers

We select separately-installed PMs that provide browser extensions in the Chrome Web Store. This selection is driven by the fact that web pages accessed through browsers serve as a representative scenario for autofill functionality, and Chrome has the highest browser market share [56]. Leveraging the ChromeStats tool [11], which has been recently utilized for a large-scale evaluation of Chrome browser extensions [19], we search for PMs with at least 100,000 (100k) active users. The search is conducted in February 29, 2024 using the keywords "password", "password-protected", and "password manager" on https://bit.ly/3B8Li4T .

TABLE 1: Selected password managers, default autofill behaviors, and the interaction strength*.

Name	Info†	Version	Personal Info				Credit Card				Login			
			Method‡	Detail	Prompt	Level♮	Method	Detail	Prompt	Level	Method	Detail	Prompt	Level
LastPass: Free Password Manager	10,000k	4.130.2.1	ClickIcon	◇	Warn	S	ClickIcon	✓	Warn	S	On load	✓	⊘	W
Avira Password Manager	6,162k	2.20.0.4570	-	-	-	-	-	-	-	-	On load	✓	⊘	W
Norton Password Manager	5,194k	8.2.0.161	-	-	-	-	ClickIcon	✓	⊘	S	ClickIcon	✓	⊘	S
1Password – Password Manager	4,443k	2.23.3	ClickIcon	×	⊘	W	ClickIcon	✓	Warn	S	ClickIcon	×	⊘	W
Bitwarden - Free Password Manager	3,903k	2024.4.1	RightClick	◇	⊘	M	RightClick	✓	⊘	S	ClickIcon	×	⊘	W
Kaspersky Password Manager	2,385k	24.0.128.1	ClickIcon[1]	✓	⊘	S	ClickIcon[1]	✓	Warn	S	On load	×	⊘	W
Dashlane — Password Manager	2,194k	6.2418.0	ClickIcon	◇	⊘	M	ClickIcon	✓	Mpw	S	On load	✓	⊘	W
iCloud Passwords	2,035k	2.2.9	-	-	-	-	-	-	-	-	ClickIcon	✓	⊘	S
Keeper Password Manager & Digital Vault	1,343k	16.8.3	RightClick	◇	Warn	S	RightClick	✓	Warn	S	ClickIcon	✓	⊘	S
MultiPassword — Password manager	1,288k	0.97.4	-	-	-	-	-	-	-	-	ClickIcon	×	⊘	W
True Key by McAfee	801k	4.3.1.9339	-	-	-	-	-	-	-	-	On load	×	⊘	W
RoboForm Password Manager	665k	9.5.9.2	ClickIcon	◇	⊘	M	ClickIcon	✓	⊘	S	ClickIcon	✓	⊘	S
DualSafe Password Manager & Digital Vault	494k	1.4.28	-	-	-	-	-	-	-	-	On load	×	⊘	W
NordPass (desktop app version)	460k	5.15.28	ClickIcon	◇	⊘	M	ClickIcon[1]	✓	⊘	S	ClickIcon	✓	⊘	S
ExpressVPN Keys: Password Manager	391k	2.0.12.715	-	-	-	-	ClickIcon	✓	⊘	S	ClickIcon	×	⊘	W
Dropbox Passwords	374k	3.26.0	-	-	-	-	ClickIcon	✓	⊘	S	On load	✓	⊘	W
KeePassXC-Browser	369k	1.9.0.4	-	-	-	-	-	-	-	-	ClickIcon	×	Warn	S
NordPass Password Manager & Digital Vault	239k	5.15.29	ClickIcon	◇	⊘	M	ClickIcon[1]	✓	⊘	S	ClickIcon	✓	⊘	S
Passbolt - Open source password manager	233k	4.7.7	-	-	-	-	-	-	-	-	ClickIcon	✓	Mpw	S
Proton Pass: Free Password Manager	210k	1.14.1	-	-	-	-	-	-	-	-	ClickIcon	✓	⊘	S
Microsoft Autofill	140k	2.0.5	ClickIcon	✓	⊘	S	ClickIcon	✓	⊘	S	ClickIcon	✓	⊘	S
Zoho Vault	134k	4.0	-	-	-	-	ClickIcon	✓	⊘	S	ClickIcon	✓	⊘	S
Enpass Password Manager	124k	6.9.3	RightClick	◇	⊘	M	ClickIcon	✓	⊘	S	ClickIcon	×	⊘	W
Password Manager SafeInCloud	107k	24.1.0	-	-	-	-	Extension	✓	⊘	S	Extension	×	⊘	W
Google Chrome	65.38%	124.0.6367.119	ClickIcon	◇	⊘	M	ClickIcon	✓	⊘	S	On load	✓	⊘	W
Microsoft Edge	12.75%	123.0.2420.81	ClickIcon	✓	⊘	S	ClickIcon	✓	⊘	S	On load	✓	⊘	W
Safari	8.72%	17.3.1	ClickIcon	✓	⊘	S	ClickIcon	✓	⊘	S	ClickIcon	✓	⊘	S
Mozilla Firefox	7.26%	125.0.3	ClickIcon	✓	⊘	S	ClickIcon	✓	⊘	S	ClickIcon	✓	⊘	S
Opera	3.05%	109.0.5097.80	ClickIcon	◇	⊘	M	ClickIcon	✓	⊘	S	On load	✓	⊘	W
Brave	-	1.65.130	ClickIcon	◇	⊘	M	ClickIcon	✓	⊘	S	ClickIcon	✓	⊘	S

∗ '-': Not applicable due to not being autofillable; ✓: Clear indication of filled form and data type; ◇: Only indication of filled form type; ×: No indication of what form and data will be filled; 'Warn': Warning dialog pops up before filling the form; 'Mpw':Master password required before filling the form; ⊘: No warnings shown, or permission and re-authentication required.
† Active users of Chrome browser extensions for 24 separately-installed PMs from ChromeStats [11] and market share of six browsers with built-in PMs sourced from StatCounter [56].
‡ Autofill triggering method. 'On load' means that the information is filled into the fields when the web page loads; 'ClickIcon' means users need to click the PM icon in the web form field or click the field to trigger the autofill functionality; 'ClickIcon[1]' means that the PM icon only appears in the *targeted sensitive* field; 'RightClick' means that users need to right-click the web page to select the PM menu to trigger the autofill functionality; 'Extension' means that users need to click the extension in the browser menu bar to trigger the autofill functionality.
♮ **W** means the PM has **W**eak user interaction strength for autofill functionality in this form, **M** for **M**edium interaction strength, and **S** for **S**trong interaction strength.

After excluding extensions with fewer than 100k active users, our screening process identifies 26 browser extensions categorized as password management software based on their descriptions in ChromeStats. Subsequently, we install each extension in a fresh Chrome browser instance to ensure that they function correctly and provide autofill functionality. During this process, we further exclude two extensions: "Delinea Web Password Filler" and "Passwordstate", as they are enterprise-focused PMs that fall outside the scope of our study. Additionally, we include the six browsers evaluated by Lin et al. [27] in our analysis. Consequently, we obtain 30 password managers (PMs) for further experiments. Detailed information about each PM is presented in Table 1.

3.2. Default Autofill Behaviors

As our research aims to investigate PMs' ability in identifying and handling hidden fields within web forms, we first explore the default autofill behaviors of PMs across various web forms to understand how PMs interact with standard web forms. Specifically, we assess whether PMs can successfully autofill each kind of form, and record the default methods for triggering the autofill functionality. This includes whether the autofill functionality presents warnings for users or requires re-authentication. Additionally, we record whether PMs prompt users with information about the categories of data being filled (e.g., password, address, phone, and credit card number). These factors indicate the strength of interaction with users, which are critical in evaluating potential real-world threats exploiting the autofill functionality without user awareness.

To ensure the autofill functionality of all PMs works well, we deploy our test website based on the autofill functionality testing website `https://fill.dev` (including various web forms), provided by one popular PM 1Password [32] with over 15 million users [1]. Then, we store fake information in the PMs for each form, and conduct experiments for each PM in a fresh browser instance generated by Selenium framework [50] to maintain a consistent initial state.

Experimental results for the default autofill behaviors are detailed in Table 1, and answers RQ1: All 30 PMs could complete the autofill process in 69 out of 90 scenarios (17 PMs for personal information forms, 22 PMs for credit card forms, and 30 PMs for login forms). There are four methods to trigger the autofill functionality: autofilling on page load (10), clicking the field or the PM icon on the form field (52), right-clicking the webpage and selecting to autofill in the web form (5), and clicking the extension icon on the browser address bar (2). PMs display different overlay styles, such as what data will be filled into the form, indicating the strength of user interaction. Based on the strength, we categorize the autofill functionality into three types ("Level" column in Table 1): (1) *Weak*, autofilling data on page load or without any information or warnings about the form type (Value "On load" in the "Method" Column or Icon × in the "Detail" column); (2) *Medium*, requiring users to click or right-click to trigger the autofill functionality, yet providing unclear information about the filled data type (Icon ◇); (3) *Strong*, providing users with clear information about form type and data type, or warnings for users (Icon ✓, or value "Warn" and "Mpw" in the "Prompt" Column).

TABLE 2: Results of our autofill testing experiments, where <input> fields are directly hidden*.

Form Type	Personal Information																Credit Card																Login Information															
Concealment Tech.†	Display: None	Visibility: Hidden	Visibility: Collapse	Opacity: 0	Covered by Overlay	Non-Effective-Size	Off-Screen	Ancestor-Overflow	Hidden	CSS Clip	CSS Clip-Path	CSS Transform	Font Size: 0	Content-Visibility	Tiny-Size		Display: None	Visibility: Hidden	Visibility: Collapse	Opacity: 0	Covered by Overlay	Non-Effective-Size	Off-Screen	Ancestor-Overflow	Hidden	CSS Clip	CSS Clip-Path	CSS Transform	Font Size: 0	Content-Visibility	Tiny-Size		Display: None	Visibility: Hidden	Visibility: Collapse	Opacity: 0	Covered by Overlay	Non-Effective-Size	Off-Screen	Ancestor-Overflow	Hidden	CSS Clip	CSS Clip-Path	CSS Transform	Font Size: 0	Content-Visibility	Tiny-Size
LastPass	×	×	✓	✓	✓	×	✓	✓	×	✓	✓	✓	×	✓	×		×[2]	×[2]	✓	×	✓	×[2]	✓	✓	×[2]	✓	✓	✓	×[2]	✓	×[2]		×	×	✓	✓	✓	×	✓	✓	×	✓	✓	✓	×	✓	×
Avira	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-	-	-	-	-	-	-	-	-	-	-	-	-	-		×	×	✓	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	✓
Norton	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		×	×	✓	×	✓	×	×	×	×	×	✓	×	✓	✓	✓		×	×	✓	×	✓	×	×	×	×	×	✓	×	✓	✓	✓
1Password	×	×	✓	✓	✓	✓	×	✓	×	×	✓	✓	✓	✓	✓		×[2]	×[2]	✓	✓	✓	✓	×[2]	✓	×[2]	×[2]	✓	✓	✓	✓	✓		×	×	✓	✓	✓	✓	×	✓	×	×	✓	✓	✓	✓	✓
Bitwarden	×	×	×	×	×	×	×	×	×	×	×	×	×	✓	×		×	×	×	×	×	×	×	×	×	×	×	×	×	✓	×		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Kaspersky	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]		×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]		×	×	×	×	×	×	×	✓	×	×	✓	✓	✓	✓	✓
Dashlane	×	×	×	×	✓	×	×	✓	×	×	✓	×	×	✓	×		×	×	×	×	✓	×	×	✓	×	×	✓	×	×	✓	×		×	×	×	×	×	×	×	✓	×	×	✓	×	×	✓	×
iCloud	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		×	×	×	✓	✓	×	×	✓	×	✓	✓	✓	✓	✓	×
Keeper	×[2]	×[2]	✓	×[2]	✓	×[2]	×[2]	✓	×[2]	✓	✓	×[2]	×[2]	✓	×[2]		×	×	✓	×	✓	×	×	✓	×	✓	✓	×	×	✓	×		×	×	✓	×	✓	×	×	✓	×	✓	✓	×	×	✓	×
MultiPassword	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		×	×	✓	×	✓	×	×	✓	×	✓	✓	×	×	✓	×
True Key	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
RoboForm	×	×	×	×	✓	×	×	×	×	×	✓	×	✓	✓	×		×[2]	×[2]	×	×	×[2]	✓	×	×[2]	×[2]	×[2]	×[2]	✓	×	✓	×	×[2]		×	×	×	×	✓	×	×	×	×	✓	×	✓	✓	×	×
DualSafe	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		×	×	×	×	✓	×	✓	✓	×	✓	✓	×	×	✓	×
NordPass(Desktop)	×	✓	✓	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	✓		×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]		×	✓	✓	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	✓
ExpressVPN Keys	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		×	×	✓	×	✓	×	×	×	×	×	✓	×	✓	✓	×		×	×	✓	×	✓	×	×	×	×	×	✓	✓	✓	✓	×
Dropbox	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		×	×	✓	×	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	✓
KeePassXC	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		×[2]	×[2]	×[2]	×[2]	×[2]	✓	×[2]	×[2]	×[2]	×[2]	×[2]	✓	✓	×[2]	×[2]	✓	×[2]
NordPass(Extension)	×	✓	✓	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	✓		×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]		×	✓	✓	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	✓
Passbolt	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Proton Pass	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		×	×	✓	✓	✓	×	×	✓	×	✓	✓	×	×	✓	×
Microsoft Autofill	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		×	×	✓	✓	✓	✓	×	✓	×	✓	✓	✓	✓	✓	✓
Zoho Vault	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		×	×	✓	×	×	×	×	×	×	✓	×	✓	×	✓	×
Enpass	×	×	✓	✓	✓	×	×	✓	×	✓	✓	×	✓	✓	✓		×	×	×	✓	×	×	×	×	×	×	×	×	✓	×	×		×	×	✓	✓	×	✓	✓	✓	×	✓	✓	✓	✓	✓	✓
SafeInCloud	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		×[2]	×[2]	×[2]	×[2]	×[2]	×[2]	×[2]	×[2]	×[2]	×[2]	×[2]	×[2]	×[2]	×[2]	×[2]		×	×	×	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	✓
Chrome	×	×	×	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	✓		×	×	×	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Edge	×	×	×	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	✓		×	×	×	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Safari	×	×	×	✓	✓	×	✓	✓	×	✓	✓	✓	✓	-[1]	×		×[2]	×[2]	×[2]	✓	✓	×[2]	✓	✓	×[2]	✓	✓	✓	✓	-[1]	×[2]		×	×	×	✓	✓	×	×	✓	×	✓	✓	✓	✓	-[1]	×
Firefox	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Opera	×	×	×	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	✓		×	×	×	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Brave	×	×	×	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	✓		×	×	×	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

* '-': Not applicable due to not autofillable; ✓: Vulnerable; ×: Not Vulnerable; -[1] means that the PM does not support the `content-visibility` property and thus the field is visible; ×[2] means that the PM fails to identify the web form type; ×[3] means that the PM places autofill-triggering fields at our examining sensitive fields, and thus they are not vulnerable to our considered threats. Note that the "Vulnerable" label means that the autofill functionality will bring security and privacy risks without considering user interaction strength. When the user interaction strength is not Weak (see the "Level" column in Table 1), users may notice that the PM is autofilling unexpected information into web forms and promptly prevent such behavior.
† Concealment techniques in 'Personal Information', 'Credit Card', and 'Login Information' columns include the left eight from Lin et al. [27] and right seven newly considered in our work.

Every password manager (PM) presents explicit information in credit card forms that this web form will be autofilled with credit card details. However, in login forms (16 out of 30) and personal information forms (10 out of 17), PMs sometimes autofill data into forms on page load (10 PMs on login forms) or display overlays without explicit information about the form type and filled data type (19, including ten PMs on personal information forms and nine PMs on login forms). For instance, the PM only displays the item name and username on the rendered overlays in the personal information form. However, it autofills sensitive information such as phone numbers and street addresses into the web form. This insufficient interaction strength poses severe privacy threats to users.

3.3. <input> Field Concealment Techniques

The effectiveness of the autofill functionality depends on the ability of password managers (PMs) to accurately identify form fields and the fillable data types. Similar to web browsers, it is also challenging for separately-installed PMs to recognize hidden fields on web forms. Beyond common CSS properties (e.g., `display:none`, `visibility:hidden`), the detection mechanism needs to consider fields covered by overlays or affected by the visibility of ancestor nodes. In this paper, we focus on the <input> field and expand upon Lin et al.'s eight concealment techniques [27] to encompass 15 techniques. We detail our newly considered techniques below, and introduce Lin et al.'s methods in Appendix A. Note that PMs may not account for the invisibility of an <input> field caused by its ancestor elements. Thus, we discuss these two cases in Section 4 and Appendix C, respectively.

HTML hidden property. The `hidden` property is an HTML standard attribute designed to conceal elements within a web page. When applied, the browser perceives these elements as non-existent, omitting them from the rendered page, and completely removing the occupied space. Although the `hidden` property does not propagate through the DOM like CSS property inheritance, its manifestation inherently occludes the visibility of all descendant elements nested within the targeted element.

CSS clip property. The `clip` property[3] facilitates the se-

3. This feature is no longer recommended and removed from web standards, but popular browsers support it partially for compatibility.

lective display of an element's specific region by *clipping* the remainder. This property is typically applied to absolutely positioned elements (i.e., setting the `position` property to `absolute` or `fixed`). By setting the `clip:rect` values, one can define the top, right, bottom, and left edges of the visible region. An element can be effectively hidden by setting a rectangle of zero size (e.g., `clip:rect(0,0,0,0)`). Like the `hidden` property, the `clip` property is non-inheritable from ancestor nodes, yet it can affect its child elements.

CSS clip-path property. The `clip-path` property replaces `clip` and enables precise control over the visible region of an element by defining a clipping path. This path delineates the portion of an element's content that should be visible, while concealing or clipping away regions outside of this path. Setting `clip-path:inset(100%)` or `clip-path:circle(0)` can conceal the element, but leave the original space available. The `clip-path` is non-inheritable, but can affect its child elements.

CSS transform property. The `transform` property allows for the visual concealment of elements through transformations, such as translation, rotation, and scaling. Elements can be proportionally enlarged or reduced in size without affecting their original layout properties. For instance, setting `transform:scale(0)` effectively hides an element by scaling it down to zero size while preserving its position in the layout. This property is not an inheritable property, but it affects its child elements.

CSS font-size property. Setting `font-size:0` renders the text content invisible while preserving the element's layout space. The `font-size` property, alongside other specified styles, such as `border` and `background`, collectively contribute to achieving the desired visual effect while maintaining the element's layout space within web pages. This property in `<input>` is inheritable from its ancestor element if no such property is specified.

CSS content-visibility property. This property controls an element's rendering and its descendants' content. Configuring this property to `hidden` conceals the element and its content, but the hidden element still occupies its original space but is not rendered. The `content-visibility` property is a non-inheritable property but has an impact on child nodes. Note that the `content-visibility` property is an experimental technology and is not yet supported on the `<input>` fields on Safari browsers. Additionally, this property is generally not used on `<input>` fields as these fields are directly rendered by browsers. In our observation, when setting `content-visibility:hidden` to `<input>` fields, the border of the`<input>` element is still visible, but the filled text is invisible. Thus, when using this property to *completely* hide an element, we need to ensure that the `<input>` element has no other styles (e.g., border or background color) to make it visible. Nevertheless, adding this property to the `<input>` element's ancestor node effectively hides this element from the web page.

Tiny size. Similar to Lin et al's *Non-Effectiveness-Size* case [27], elements with tiny size (e.g., width=1px) can be concealed from the webpage. Note that the width of the `<input>` element can be influenced by its ancestor node's width only when its width is set using a relative length unit (e.g., em), which means the `<input>` element's width also changes. If the `<input>` element's width is set with a specific number, it will not be influenced by its ancestor's width. Thus, we regard this case the same as the case where the `<input>` field is directly configured. We only consider the influence of this concealment technique and *Non-Effectiveness-Size* technique on the `<input>` element instead of its ancestor nodes.

We first explore cases where only the `input` field is configured as invisible without modifying the CSS property of its ancestor element, and report results in Table 2. Then, we report cases that `input`'s ancestor element is configured as invisible using CSS property, making the `input` element invisible from the web page. For ancestor elements, we drop two cases that modify the width of the ancestor element to a non-effective size or tiny size. This is because the width of `<input>` field will be influenced by its ancestor element only when it is set as relative length, which also means the width of the `<input>` element is changed. Finally, we discuss experimental results in appendix C.

3.4. Hidden fields autofilling in web forms

This paper considers password fields in login forms, phone and address fields in personal information forms, and credit card number fields in credit card forms as sensitive information. We utilize the above 15 concealment techniques to hide these fields from web pages. Then, we trigger PMs to autofill corresponding web forms and record whether hidden fields are filled with proper data.

We have identified 69 autofill scenarios in Sec. 3.2, requiring us to trigger a total of 1,032 autofill operations to fill forms (=69×15-3, where 15 refers to fifteen concealment techniques for `<input>` fields[4], and three for Safari's lack of support for the `content-visibility` property). Manually verifying whether the data is filled into hidden fields is inconvenient, as it necessitates opening browser developer tools and executing commands like `document.getElementById("#id").value` to retrieve the filled data. Thus, we implement a *semi-automated* tool using Selenium [50] to automate interactions with browser extensions, capture screenshots, and retrieve filled data, making our analysis faster and more repeatable. Note that PM extensions may require users to log in before using the autofill functionality, which requires manual intervention. Additionally, operations like clicking the extension icon in the browser menu bar to trigger the autofill functionality are inconvenient for end-to-end testing tools like Selenium [50], Puppeteer [13], and PlayWright [33]. Consequently, we manually trigger the autofill functionality and conduct experiments for these specific cases.

We modify the autofill functionality testing form provided by 1Password [32] at `https://fill.dev/` to

4. Incorporating the 13 concealment techniques for the ancestor elements of `<input>` elements, we need to trigger 1,926 autofill operations.

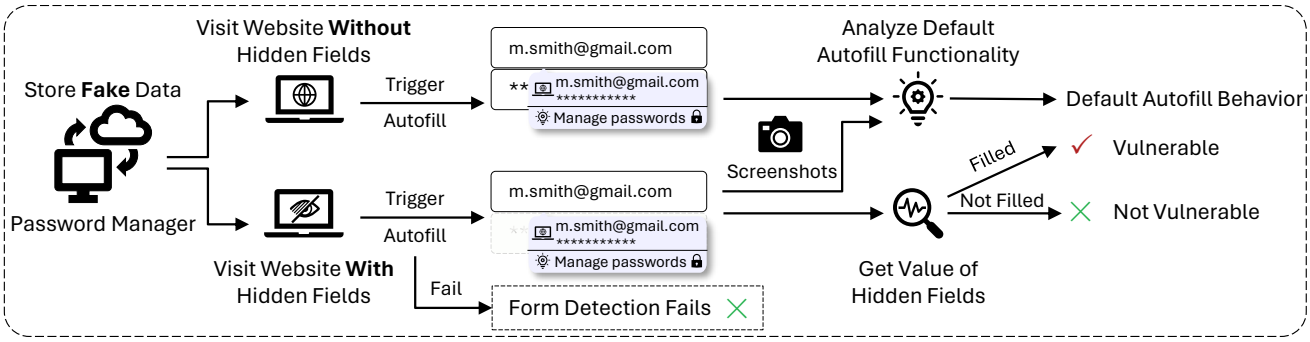


Figure 3: Our experimental process on the autofill functionality's ability to detect and handle hidden fields on web forms.

accommodate our testing scenarios. Specifically, we use the source code of our tested three web forms from this website, apply various concealment techniques (i.e., CSS properties or HTML properties) to hide sensitive `<input>` fields, and create testing webpages. The testing process is detailed in Figure 3. We initially store corresponding fake data in the PM. Then, we manually check the autofill functionality in a *standard* web form without any hidden fields, to ensure it works well in standard forms. We record the default autofill functionality trigger method and the style of pop-up overlays. Subsequently, we utilize the semi-automated tool to systematically test each concealment technique across PMs and record whether data is filled into hidden fields. Additionally, PMs such as Safari and Firefox will prompt users with ready-to-fill data type in personal information forms (such as "Also autofill address, phone" in Firefox). We also record the pop-up overlays on web forms with hidden fields. As expected, if PMs provide filled data type details like Firefox and Safari, they will change the prompts when hidden fields are not filled. We have provided the artifact of our work [6], including the source code of our semi-automated tool and the testing websites, and 24 PM extensions in the Chrome browser used in our experiments, associated with detailed documents.

4. Experimental results for autofill

We now delve into our experimental findings as detailed in Table 2. We specifically focus on the results when the `<input>` field is hidden due to its properties. Results of ancestor elements' impact are presented in Appendix C.

We aim to ascertain whether significant disparities exist in the filled probability when encountering hidden fields concerning: (1) the PM type (separately-installed (Extension) and built-in-browser (Browser) PMs), (2) the web form type (login (Login) forms, personal information (PII) forms, and credit card (CVV) forms), and (3) 15 concealment techniques. Since these analyses entail comparing proportions, we employ the χ^2 test to investigate whether significant differences exist among them. To account for multiple comparisons, we apply the Bonferroni correction, adjusting alpha levels to 0.017 per test (=0.05/3). Additionally, we also provide the filled results of different PM types, web form types, and concealment techniques in Figures 4 and 5. The detailed filled probabilities are presented in Tables 4 and 5 in Appendix D. Our analysis addresses RQ2.

4.1. Differences between PM types

We test the following hypothesis to explore differences between separately-installed PMs and built-in-browser PMs. H_0: The filled probability has no significant relationship with the type of the password manager; H_a: The filled probability has a significant relationship with the type of the password manager.

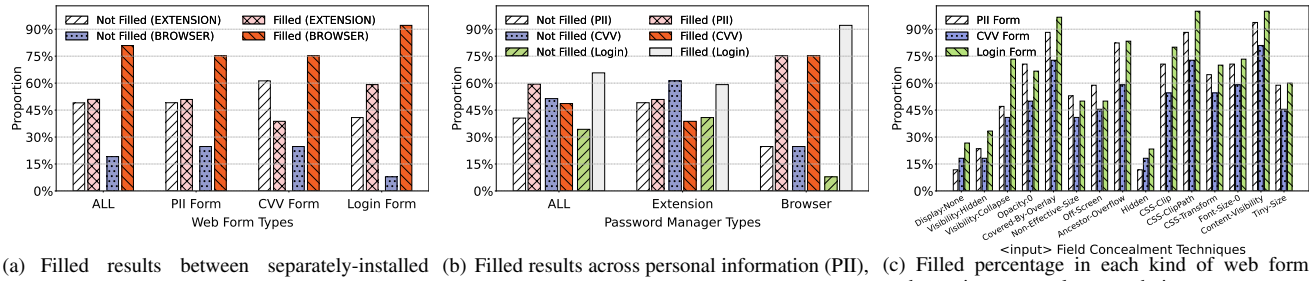
We conduct a χ^2 test and reject the null hypothesis H_0 (χ^2=71.859, p<0.01), indicating a significant relationship between the filled probability and the PM type. The effect size is weak (Cramer's V=0.266). Therefore, we accept our alternative hypothesis H_a that the filled probability differs between separately-installed PMs and built-in-browser PMs. Subsequently, we perform a binary logistic regression test to assess the impact of various PM types. Results show that built-in-browser PMs are significantly more likely to fill in hidden fields compared to separately-installed PMs. To provide a more intuitive explanation, we exponentialize these coefficients, and obtain the odd ratios: As the number of testing samples increases, the filled probability of built-in-browser PMs increases by approximately 4.07 (=(216/51)/(390/375), see Tab. 4 for more details) times compared to separately-installed PMs. Moreover, as shown in Figure 4(a), built-in-browser PMs are more likely to fill in hidden elements regardless of whether we consider the web form type and which form type we consider.

Takeaway. Built-in-browser PMs are more likely (∼4.07 times) to fill hidden fields than separately-installed PMs.

4.2. Differences between web form types

We test the hypothesis to explore differences among three web forms. H_0: The filled probability has no significant relationship with the web form type; H_a: The filled probability has a significant relationship with the web form type.

We conduct a χ^2 test and reject the null hypothesis H_0 (χ^2=22.897, p<0.01). The effect size is weak (Cramer's V=0.149). We accept our alternative hypothesis H_a that the filled probability differs among the tested three web forms. Then, we conduct logistic regression tests to evaluate that



(a) Filled results between separately-installed (Extension) and built-in-browser (Browser) PMs.

(b) Filled results across personal information (PII), credit card (CVV), and login (Login) forms.

(c) Filled percentage in each kind of web form under various concealment technique.

Figure 4: Analysis for experimental results when <input> elements are invisible due to their own property.

hidden fields in which web forms are more likely to be filled. Our results show that hidden fields in credit card forms are significantly less likely to be filled compared to login forms, with odd ratios of 0.494. However, there is no significant difference between personal information and either credit card forms or login forms. Though not statistically significant, the odd ratio between personal information forms and login forms is 0.765, indicating that hidden fields in login forms are more likely to be filled in. In Figure 4(b), login forms have the highest filled probability. For personal information forms, two kinds of PMs autofill data into hidden fields with a percentage of>50%. For credit card forms, separately-installed PMs perform better (the filled probability<50%).

In addition to the relationship between filled probability and web form types, not every PM supports the autofill functionality for any of the three web forms: 17 PMs support all web forms, 22 PMs support credit card forms and login forms, and all 30 PMs support login forms. PMs capable of filling at least two web forms may perform differently when identifying hidden elements in various web forms. Excluding scenarios where autofill functionality is triggered in the sensitive field, only Firefox, Norton, 1Password, Dashlane, Keeper, ExpressVPN, and two NordPass extensions behave consistently when facing different web forms. This indicates that it is necessary to conduct experiments for each type of web form. Additionally, among the six browser-based PMs, only Safari could detect seven concealment techniques in login forms, while the others fill passwords into hidden fields concealed using each of the 15 techniques.

Takeaway. Hidden fields in credit card forms are significantly less likely (0.494 times) to be filled than login forms. There is no significant difference among other forms.

4.3. Differences between concealed techniques

We test the following hypothesis for differences between fifteen concealment techniques. H_0: The filled probability has no significant relationship with the used concealment technique; H_a: The filled probability has a significant relationship with the used concealment technique.

We conduct a χ^2 test and reject the null hypothesis H_0 (χ^2=216.89, p<0.01). The effect size is moderate (Cramer's V=0.458). Therefore, we accept our alternative hypothesis H_a that the filled probability differs among fifteen concealment techniques. We then employ a decision tree analysis to investigate the influence of concealment techniques on filled probability. When <input> fields are hidden using `display:none`, `visibility:hidden`, or `hidden`, the filled probability is 21.74% (=45/207, see Table 5), indicating PMs properly consider the three techniques. Specifically, for separately-installed PMs, the filled probability is only 15.69% under these three techniques. For built-in-browser PMs, the filled probability is 16.7% for personal information and credit card forms, yet 83.3% for login forms. This indicates that built-in-browser PMs have insufficient consideration about hidden fields in login forms.

For the other 12 techniques, the filled probability is 68% (=561/825), which is relatively higher than the above three techniques, indicating PMs have inadequate consideration for these concealment techniques. The filled probability for built-in-browser PMs is 91.5%. For separately-installed PMs, properties `covered-by-overlay`, `clip-path`, and `content-visibility` have a higher probability of 85.6%. More specifically, separately-installed PMs fill in hidden fields in credit card forms with 36.1% probability. Besides, under `off-screen`, `non-effective-size`, and `tiny-size` techniques in personal information forms and login forms, separately-installed PMs could detect the hidden fields and would not autofill with 43% probability.

As shown in Figure 4(c), most PMs successfully detect three specific concealment techniques: `display:none`, `visibility:hidden`, and HTML `hidden` properties, while lacking robust detecting methods for other 12 techniques, particularly for login forms, where nine techniques evade $\geqslant$ 20 PMs. Besides, no techniques could deceive all considered PMs in all kinds of web forms, and there are no techniques detected by every PM in any web form.

Takeaway. 83.3% (=25/30) PMs successfully detect three out of the 15 concealment techniques (`display:none`, `visibility:hidden`, and HTML `hidden`) in at least one form, but lack robust methods for other 12 techniques.

4.4. Improvements of built-in-browser PMs

Compared to Lin et al.'s study [27], there are *no significant* improvements in identifying and handling hidden elements for built-in-browser PMs. These PMs, except Firefox, only avoid filling data into hidden fields concealed by `display` and `visibility` CSS property, as well as the

HTML `hidden` property in personal information and credit card forms, aligning with Lin et al.'s findings (see Table 1 in [27] for more details). Only Safari has enhanced its ability to detect elements of non-effective size or tiny-size and regard them as invisible. However, these PMs perform poorly in login forms, with only Safari detecting and stopping seven concealment techniques. The other five PMs do not consider handling hidden fields in login forms, potentially posing severe risks to users' credentials. Assisted with the analysis of the differences between built-in-browser and separately-installed PMs in Section 4.1, we answer RQ3.

4.5. Case Studies on Open-Sourced PMs

We present case studies for three open-source, separately-installed PMs and two built-in-browser PMs, exploring their element visibility checking mechanism, including Bitwarden [7], Passbolt [47], KeePassXC [23], Chromium-based browsers [17], and Firefox [36].

1) Bitwarden: Bitwarden [7] does not autofill any sensitive data in personal information and credit card forms except for the `content-visibility` technique, but it autofills passwords into hidden fields under any techniques. We conduct static and dynamic analysis on the source code of the Bitwarden browser module [16]: Bitwarden develops three functionalities to check if a form field is viewable, including whether the element is within the viewport bounds, hidden by CSS, and not hidden behind another element.[5]

Despite autofilling passwords into hidden fields, Bitwarden can detect hidden password fields in login forms. Specifically, Bitwarden detects visible password fields first. If no visible password fields are found, it searches for hidden password fields[6] on the web page. Once found, Bitwarden will also autofill passwords into hidden fields. We report this issue to Bitwarden. They respond that their practice is for performance and usability considerations. They also claim that two-step authentication may require passwords to be autofilled even if the password fields are invisible.

2) Passbolt: Passbolt [47] is relatively secure when conducting autofill functionality, as shown in Table 1. It only supports autofill functionality in login forms, presenting clear prompts for the form type and requiring a master password before autofilling by default, indicating strong interaction with users. However, Passbolt autofills passwords into hidden password fields concealed by 14 out of 15 techniques (only detecting and handling HTML `hidden` property). For ancestor nodes' influence, Passbolt even autofills passwords into hidden fields concealed by any of the considered 15 techniques. Our static analysis of the source code reveals that Passbolt does not implement a proper mechanism to detect hidden fields.

3) KeePassXC: KeePassXC [23] also only provides autofill functionality for login forms, and we delve into its source code to explore the visibility mechanism. KeePassXC has implemented a mechanism to check whether the element is within the web view. Additionally, it also checks the `visibility` and `opacity` of this element. However, it does not consider the elements hidden by newly considered techniques such as `clip` and `clip-path` and the influence of ancestor nodes. We provide methods to improve the element visibility detection mechanism [23] to KeePassXC developers. However, they respond that a comprehensive visibility check may be very slow and costly, and their practice is a compromise to quick element visibility check without slowing down the extension.

4) Chromium-based browsers: PMs built in Chromium-based browsers (i.e., Chrome, Edge, Brave, and Opera) autofill passwords into hidden fields regardless of the concealment technique. For personal information and credit card forms, PMs could detect fields concealed by `display`, `visibility`, and `hidden` properties.

We search for the visibility check function in Chromium source code [17]. During our static analysis, we identify a "IsWebElementVisible" function in "form_autofill_util.cc"[7], which detects the visibility of web elements. This function checks whether a `<input>` element's dimensions and scroll size are greater than or equal to 10 pixels. This method primarily relies on inspecting the size of the element to check the visibility. Thus, this function could check elements concealed by `visibility`, `display`, and `hidden` properties. However, other field concealment techniques are more challenging to detect with this function.

5) Firefox browser: We conduct a static analysis on the source code of Firefox browser [36]. The `fillFields` function in "FormAutofillHandler.sys.mjs" is responsible for populating autofillable form elements. Before conducting autofilling, this function invokes the "isFieldAutofillable" function located in "FormAutofillUtils.sys.mjs", which primarily checks whether the element exists, is not marked as `readonly`, and is not `disabled`. Although the second file defines a function named `isFieldVisible`, which evaluates the element's opacity and `visibility` CSS property, this function is not utilized in the autofill process.

Our static analysis concludes that Firefox's autofill functionality does not perform element visibility checks when autofilling form fields. This finding is consistent with our experimental results: Firefox autofills data into hidden fields regardless of the concealment technique.

4.6. Responsible Disclosure

We responsibly disclose the privacy threats we identify to PM operators or developers. Several PMs have responded to our disclosures. LastPass [25] has confirmed this issue but has not resolved it. 1Password [32] responds that a compromised client-side is not considered in their security model. They believe that once the user chooses to use the autofill functionality, the burden of ensuring security rests on the users. Other PMs, such as KeePassXC and Bitwarden, respond that their implementation primarily considers the balance between user experience, performance, and security.

5. The `isFormFieldViewable` functionality in the source code can be found in https://bit.ly/3Wekxop in Bitwarden Github repository.

6. The `generateLoginFillScript` functionality in the source code can be found in https://bit.ly/4bebjwB in Bitwarden Github repository.

7. Located in "chromium/src/components/autofill/content/renderer/".

5. Countermeasures

PMs could address autofill issues by considering various concealment techniques to ensure only visible fields are filled. However, as several PMs have noted, such comprehensive considerations may introduce performance issues and suboptimal user experiences. Gautam et al. [14] recently proposed countermeasures against malicious client-side attackers aiming to steal PM-filled data: One effective method involves PMs autofilling nonces and relying on browsers to replace them with passwords before sending web requests. However, this approach requires browser modifications. Besides, non-PM users who fill forms manually would not be protected by this method, as it requires PMs to register nonces initially and later autofill data into web forms.

We provide countermeasures to mitigate potential threats brought by the autofill functionality from two aspects.

1) *Increasing user interaction strength.* When PMs provide strong user interaction during autofilling, such as providing detailed information about what form and data to be filled, providing users with clear warnings or alerts, or preventing autofill on page load, issues revealed in this paper are unlikely to pose significant privacy threats. Particularly, displaying the form type and filled data type before autofilling is a practical approach to prevent users from filling data into hidden fields. Thus, this method could enhance the security of PMs and avoid information leakage.

This practice is implemented in 66.7% (=20/30) PMs for login forms and all PMs for credit card forms. However, seven out of the 20 PMs in login forms autofill credentials on page load, which means that users may not notice the rendered overlay that provides detailed information about the form type. In personal information forms, only Firefox and Safari notify users of the data type to be filled in (e.g., "Autofill phone, address" in Safari). Edge browser and Microsoft Autofill differentiate the information in rendered pop-up overlays when the hidden fields are not filled. Although these PMs behave better, as pointed out by Lin et al. [27], the data type is presented at a coarse-grained level. We also observe the same type of rendered overlay. For example, Safari prompts "Autofill address" when it detects the "City" field as visible. Thus, PM operators could provide detailed and fine-grained prompts in the rendered overlay for the data type to be filled into web forms.

2) *A new perspective of countermeasure based on visual language model (VLM).* Considering the emergence of new HTML or CSS properties that can be used to hide elements from the web view (though not designed for this purpose), inspecting the web page source code is a feasible but complex approach, and may require continuous updates. We propose a new perspective based on VLM to help PMs avoid filling data into hidden fields. A browser extension or other tools with screenshot permissions can capture the screenshots of web forms, and then utilize the VLM technique to identify the visible form fields and analyze the data type to be filled. For example, using an open source VLM CogVLM [64], PMs can identify fillable elements within web pages and infer data types by providing prompts like "Provide the number of web form input fields and the filled data type" to the model.

Though the performance is relatively poor in the web scenarios (about 18 seconds to handle a request using the provided API [38]), the VLM technique is useful for detecting fillable fields. Despite being highly resource-intensive, which may further influence the usage rate of this functionality [46], a dedicated model can more effectively complete our task. Furthermore, Google has deployed an image classifier to detect phishing websites, we believe countermeasures based on VLM are practical and could assist the visibility check method of PMs.

6. Discussion and Future Work

We now discuss the differences with Lin et al.'s work [27], and provide the limitations and possible future work.

Differences between our work and Lin et al.'s work. While our study and Lin et al.'s work [27] analyze the privacy threats posed by the autofill functionality, our work differs from it in three aspects. First, their work primarily considers built-in-browser PMs, whereas our study primarily focuses on separately-installed PMs, which are perceived to be more secure [30], [48]. Furthermore, we extend our analysis to compare the hidden field detection capabilities of these two types of PMs. Second, while both studies consider hidden fields in personal information and credit card forms, our research investigates credential theft risks in login forms. Third, we expand their considered eight concealment techniques to 15 techniques, including new techniques like `clip-path` and `content-visibility`, which effectively hide fields but are not detected by PMs.

Limitations and Future Work. 1) Limited consideration of concealment techniques: The fifteen concealment techniques presented in Section 3.3 are either heuristic (7) or introduced from Lin et al.'s work (8) [27]. Besides, for advanced techniques such as `clip-path`, we only implement one common property (i.e., `clip-path:inset`), and do not test other properties (e.g., `clip-path:circle`) that can be used to conceal elements. This indicates that our tests may overlook the potentially effective concealment techniques. Still, the privacy threats we have identified remain significant, as new techniques may not be well considered by PM operators (e.g., the `content-visibility` technique is proposed in 2020, and all PMs fill passwords into hidden `<input>` fields concealed by this method). We believe it is necessary to propose a comprehensive and effective detection mechanism for emerging techniques.

2) Only focus on the security perspective: Our paper primarily discusses the autofill functionality from a security perspective, emphasizing the privacy threats posed by filling data into hidden fields. However, as responded by PM operators, their practices balance performance, usability, and security. Comprehensive checks for autofill functionality may damage user experience. A user survey is necessary to explore users' perceptions of autofilling data into hidden fields and whether security/privacy risks prevent users from using the autofill functionality. We leave this as future work.

7. Related Work

We first review the research associated with the autofill functionality of password managers (PMs). Then, we explore recent studies concerning the security and privacy threats posed by malicious browser extensions.

7.1. Password Manager's Autofill Functionality

We have summarized several works about the security and privacy threats brought by the autofill functionality of built-in-browser PMs [4], [27], [57] in Section 1, particularly when facing hidden fields in web forms. Additionally, previous research has examined the security and usability of PMs' autofill functionality. At IEEE S&P'21, Huaman et al. [20] utilized public user reviews from Chrome browser extensions and Github issues to investigate the interaction issues between PMs and browsers. They revealed multiple issues related to the autofill functionality, such as filling web forms with additional elements (e.g., one-time-password) and across different subdomains. Their work considered hidden elements (see J-01 in Table III in their work [20]) but focused only on the `display:none` technique and the potential effects on user interaction experience.

At ACSAC'21, Simmons et al. [53] systematized 17 PM use cases. They discussed three aspects of autofill functionality: whether the autofill functionality requires user interaction, autofill on different domains, and whether autofill requires a trusted pathway. They revealed that 50% of desktop PMs and 8% of mobile PMs do not require user interaction to autofill (i.e., weak user interaction as shown in Table 1), which may bring severe privacy threats to users, especially when hidden fields are filled.

At USENIX SEC'20, Oesch and Ruoti [41] conducted a security evaluation on the autofill functionality, including whether autofill requires user interaction, fills data into iframes, fills forms that differ from the environment of saved forms, and fills data into non-standard login fields. Besides, they mentioned potential mitigation to protect autofill from XSS attacks proposed by Stock and Johns [57]. This technique uses random nonces when filling forms and replacing the values with passwords when transferring the data. Recently, Gautam et al. [14] implemented an improved design of this method to prevent malicious client-side scripts and browser extensions from stealing filled data.

At USENIX SEC'14, Silver et al. [52] discussed the threats of autofill functionality of four PMs, which are susceptible to sweep attacks where attackers inject JavaScript code to webpages to retrieve passwords without user interaction. They also explored whether PMs autofill web forms when the `<input>` field is invisible but only considering `display:none` and `opacity:0`. At the same conference, Li et al. [26] conducted a security analysis of five PMs, including attack scenarios where the autofill functionality is triggered. In contrast, our main idea is not mitigating web vulnerabilities such as XSS attacks, but highlights the privacy threats brought by such autofill functionality, which may autofill sensitive data into hidden fields.

7.2. Threats brought by Malicious Extensions

Recent studies [10], [44], [54], [55] have extensively discussed security and privacy issues stemming from malicious browser extensions, including credential theft, user tracking, advertisement injection, and remote code execution. Despite efforts by web stores to implement security check mechanisms for extensions, well-designed malicious extensions can often bypass the security checks, posing significant security risks to users. Nayak et al. [39] identified that malicious extensions could access filled password values in login web pages and even pass through the Chrome web store security check. At EuroS&P'23, Miguel et al. [34] explored security issues arising from extensions exploiting Chrome DevTools Protocols, such as stealing user information, monitoring network traffic (including the transferred credentials), and bypassing security interstitial pages. Besides, at IEEE S&P'22, Agarwal et al. [5] highlighted the risk of malicious extensions accessing credential stores in PM extensions directly, driving the deployment of the extension isolation mechanism in Chrome browsers. Thus, it is now difficult for a malicious extension to directly access the content of PM extensions.

In this paper, we consider attackers who can inject hidden fields into web pages which only require limited privileges. Note that our focus is on exploiting weaknesses in the autofill functionality for handling hidden fields, that could make data inadvertently be filled into hidden fields, bringing severe privacy threats to users.

8. Conclusions

We have conducted the *first* empirical study on the effectiveness of separately-installed password managers (PMs) in detecting and handling hidden fields in web forms. We reveal that leading PMs cannot effectively detect and handle hidden fields concealed by twelve out of fifteen concealment techniques. The most concerning is the login forms, which have the highest filled probability 65.7% among the three forms, and all 30 PMs autofill passwords into hidden fields concealed by `content-visibility` and `clip-path` properties. Coupled with the prevalence of weak user interaction prompts during autofill (e.g., autofilling on page load or providing nothing about the filled data type), these vulnerabilities pose serious privacy threats to users. We hope our findings and suggestions from the security perspective could help PM operators improve their autofill functionality, and better balance its security and usability as users are more and more concerning their online privacy rights.

Acknowledgment

We appreciate anonymous reviewers for invaluable comments. Ding Wang is the corresponding author. This research was in part supported by the National Natural Science Foundation of China under Grants Nos. 62222208 and 62172240, and by the Fundamental Research Funds for the Central Universities, Nankai University (Grant No. 63243154).

References

[1] *1Password's Blue Ocean Strategy*, 2022, https://bit.ly/482fYRR.

[2] *Department of Justice Statement on the intrusion into the Department's Microsoft O365 email environment*, Oct. 2022, https://bit.ly/40muAHY.

[3] G. Acar, *No boundaries for user identities: Web trackers exploit browser login managers*, Dec. 2017, https://bit.ly/3JYGj8h.

[4] G. Acar, S. Englehardt, and A. Narayanan, "No boundaries: data exfiltration by third parties embedded on web pages," in *Proc. PETS 2020*, pp. 220–238.

[5] A. Agarwal, S. O'Connell, J. Kim, S. Yehezkel, D. Genkin, E. Ronen, and Y. Yarom, "Spook.js: Attacking Chrome Strict Site Isolation via Speculative Execution," in *Proc. IEEE S&P 2022*, pp. 699–715.

[6] L. Autofill, *LeakyAutofill-Artifact*, Sep. 2024, https://bit.ly/4gQe9eN.

[7] Bitwarden, *Bitwarden Security Whitepaper*, 2024, https://bitwarden.com/help/bitwarden-security-white-paper/.

[8] Canada.ca, *Password Guidance*, 2018, https://bit.ly/47dTBIf.

[9] CCPA, *California Consumer Privacy Act (CCPA)*, Mar 2024, https://oag.ca.gov/privacy/ccpa.

[10] Q. Chen and A. Kapravelos, "Mystique: Uncovering information leakage from browser extensions," in *Proc. ACM CCS 2018*.

[11] Chrome-Stats, *Chrome-Stats*, 2024, https://chrome-stats.com/.

[12] CISA, *Choosing and Protecting Passwords*, Nov. 2019, https://bit.ly/3tMqwEO.

[13] G. Developer, *Puppeteer*, 2024, https://pptr.dev/.

[14] A. Gautam, T. K. Yadav, K. Seamons, and S. Ruoti, "Passwords Are Meant to Be Secret: A Practical Secure Password Entry Channel for Web Browsers," *arXiv preprint arXiv:2402.06159*, 2024.

[15] GDPR, *Art. 5 GDPR Principles relating to processing of personal data*, 2024, https://gdpr.eu/article-5-how-to-process-personal-data/.

[16] B. Github, *Bitwarden Clients App Browser*, 2024, https://github.com/bitwarden/clients/tree/main/apps/browser.

[17] Google, *Chromium Code Search*, 2024, https://source.chromium.org/.

[18] A. Hanamsagar, S. S. Woo, C. Kanich, and J. Mirkovic, "Leveraging semantic transformation to investigate password habits and their causes," in *Proc. CHI 2018*, pp. 1–12.

[19] S. Hsu, M. Tran, and A. Fass, "What is in the Chrome Web Store?" in *Proc. ACM AsiaCCS 2024*, pp. 1–14.

[20] N. Huaman, S. Amft, M. Oltrogge, Y. Acar, and S. Fahl, "They would do better if they worked together: The case of interaction problems between password managers and websites," in *Proc. IEEE S&P 2021*.

[21] A. Hutchinson, J. Tang, A. J. Aviv, and P. Story, "Measuring the Prevalence of Password Manager Issues Using In-Situ Experiments," in *Proc. USEC 2024*, pp. 1–27.

[22] I. Ion, R. Reeder, and S. Consolvo, ""...No one Can Hack My Mind": Comparing Expert and Non-Expert Security Practices," in *Proc. SOUPS 2015*, pp. 327–346.

[23] KeePassXC, *KeePassXC Browser Extension*, 2024, https://github.com/keepassxreboot/keepassxc-browser.

[24] Lastpass, *The 3rd Annual Global Password Security Report*, 2019, https://bit.ly/4eSx2ff.

[25] LastPass, *LastPass Technical Whitepaper*, Aug. 2023, https://bit.ly/4eTB9aU.

[26] Z. Li, W. He, D. Akhawe, and D. Song, "The Emperor's New Password Manager: Security Analysis of Web-based Password Managers," in *Proc. USENIX SEC 2014*, pp. 465–479.

[27] X. Lin, P. Ilia, and J. Polakis, "Fill in the blanks: Empirical analysis of the privacy threats of browser form autofill," in *Proc. ACM CCS 2020*, pp. 507–519.

[28] S. G. Lyastani, M. Schilling, S. Fahl, M. Backes, and S. Bugiel, "Better managed than memorized? Studying the Impact of Managers on Password Strength and Reuse," in *Proc. USENIX SEC 2018*.

[29] K. Lyons, *Hackers reportedly used a compromised password in Colonial Pipeline cyberattack*, June 2021, https://bit.ly/40kIsCw.

[30] P. Mayer, C. W. Munyendo, M. L. Mazurek, and A. J. Aviv, "Why Users (Don't) Use Password Managers at a Large Educational Institution," in *Proc. USENIX SEC 2022*, pp. 1849–1866.

[31] P. Mayer, Y. Zou, F. Schaub, and A. J. Aviv, "" Now I'm a bit angry:" Individuals' Awareness, Perception, and Responses to Data Breaches that Affected Them." in *Proc. USENIX SEC 2021*, pp. 393–410.

[32] P. Memberships, *1Password Security Design*, Oct. 2023, https://1passwordstatic.com/files/security/1password-white-paper.pdf.

[33] Microsoft, *Playwright enables reliable end-to-end testing for modern web apps*, 2024, https://playwright.dev/.

[34] J. M. Moreno, N. Vallina-Rodriguez, and J. Tapiador, "Chrowned by an Extension: Abusing the Chrome DevTools Protocol through the Debugger API," in *Proc. EuroS&P 2023*, pp. 832–846.

[35] Mozilla, *Fathom*, 2019, https://mozilla.github.io/fathom/.

[36] mozilla, */ - mozsearch*, 2024, https://bit.ly/4exskDX.

[37] C. W. Munyendo, P. Mayer, and A. J. Aviv, ""I just stopped using one and started using the other": Motivations, Techniques, and Challenges When Switching Password Managers," in *Proc. ACM CCS 2023*.

[38] naklecha, *CogVLM API*, 2024, https://bit.ly/4esGOFl.

[39] A. Nayak, R. Khandelwal, E. Fernandes, and K. Fawaz, "Experimental Security Analysis of Sensitive Data Access by Browser Extensions," in *Proc. WWW 2024*, pp. 1–12.

[40] NIST, "NIST SP 800-63B digital identity guidelines: Authentication and lifecycle management," Tech. Rep., June 2017.

[41] S. Oesch and S. Ruoti, "That Was Then, This Is Now: A Security Evaluation of Password Generation, Storage, and Autofill in Browser-Based Password Managers," in *Proc. USENIX SEC 2020*.

[42] B. Pal, T. Daniel, R. Chatterjee, and T. Ristenpart, "Beyond credential stuffing: Password similarity models using neural networks," in *Proc. IEEE S&P 2019*, pp. 417–434.

[43] S. Pearman, J. Thomas, P. E. Naeini, H. Habib, L. Bauer, N. Christin, L. F. Cranor, S. Egelman, and A. Forget, "Let's Go in for a Closer Look: Observing Passwords in Their Natural Habitat," in *Proc. ACM CCS 2017*, pp. 295–310.

[44] P. Picazo-Sanchez, B. Eriksson, and A. Sabelfeld, "No Signal Left to Chance: Driving Browser Extension Analysis by Download Patterns," in *Proc. ACSAC 2022*, pp. 896–910.

[45] H. Ray, F. Wolf, R. Kuber, and A. J. Aviv, "Why older adults (Don't) use password managers," in *Proc. USENIX SEC 2021*, pp. 73–90.

[46] G. S. Research, *The probability of bounce increases 32% as page load time goes from 1 second to 3 seconds*, 2017, https://bit.ly/3ytGohR.

[47] P. SA, *Passbolt Browser Extension*, 2022, https://github.com/passbolt/passbolt_browser_extension.

[48] P. Sarah, Z. Shikun, B. Lujo, and C. Nicolas, "Why people (don't) use password managers effectively," in *Proc. SOUPS 2019*, pp. 319–338.

[49] S. Seiler-Hwang, P. A. Cabarcos, A. Marín, F. Almenáres, D. D. Sánchez, and C. Becker, ""I don't see why I would ever want to use it": Analyzing the Usability of Popular Smartphone Password Managers," in *Proc. ACM CCS 2019*, pp. 1937–1953.

[50] Selenium, *The Selenium Browser Automation Project*, 2024, https://selenium.dev/documentation/.

[51] A. Senol, G. Acar, M. Humbert, and F. Z. Borgesius, "Leaky Forms: a study of email and password exfiltration before form submission," in *Proc. USENIX SEC 2022*, pp. 1813–1830.

[52] D. Silver, S. Jana, D. Boneh, E. Chen, and C. Jackson, "Password managers: Attacks and defenses," in *Proc. USENIX SEC 2014*.

[53] J. Simmons, O. Diallo, S. Oesch, and S. Ruoti, "Systematization of Password ManagerUse Cases and Design Paradigms," in *Proc. ACSAC 2021*, pp. 528–540.

[54] A. Sjosten, S. V. Acker, P. Picazo-Sanchez, and A. Sabelfeld, "Latex Gloves: Protecting Browser Extensions from Probing and Revelation Attacks," in *Proc. NDSS 2019*, pp. 1–15.

[55] D. F. Somé, "EmPoWeb: Empowering Web Applications with Browser Extensions," in *Proc. IEEE S&P 2019*, pp. 227–245.

[56] S. G. Stats, *Desktop Browser Market Share Worldwide*, 2024, https://gs.statcounter.com/browser-market-share/desktop/worldwide.

[57] B. Stock and M. Johns, "Protecting users against XSS-based password manager abuse," in *Proc. ASIACCS 2014*, pp. 183–194.

[58] S. Team, *Password Manager Industry Report and Market Outlook (2023-2024)*, Sep. 2023, https://bit.ly/3LVdB9A.

[59] A. Titterington, *Dangerous browser extensions*, Dec. 2023, https://bit.ly/3TR0ZEj.

[60] K. Viezelyte, *Juggling security: How many passwords does the average person have in 2024?*, Apr. 2024, https://bit.ly/47YkyAr.

[61] D. Wang, P. Wang, D. He, and Y. Tian, "Birthday, name and bifacial-security: Understanding passwords of Chinese web users," in *Proc. USENIX SEC 2019*, pp. 1537–1555.

[62] D. Wang, Z. Zhang, P. Wang, J. Yan, and X. Huang, "Targeted online password guessing: An underestimated threat," in *Proc. ACM CCS 2016*, pp. 1242–1254.

[63] D. Wang, Y. Zou, Y.-A. Xiao, S. Ma, and X. Chen, "Pass2Edit: A Multi-Step Generative Model for Guessing Edited Passwords," in *Proc. USENIX SEC 2023*, pp. 983–1000.

[64] W. Wang, Q. Lv, W. Yu, W. Hong, J. Qi, Y. Wang, J. Ji, Z. Yang, L. Zhao, X. Song *et al.*, "Cogvlm: Visual expert for pretrained language models," *arXiv preprint arXiv:2311.03079*, 2023.

[65] web.dev, *Autofill*, Dec. 2021, https://web.dev/learn/forms/autofill.

[66] O. Williams, *This could be the iCloud flaw that led to celebrity photos being leaked*, Sep. 2014, https://bit.ly/47bwflO.

Appendix

A. Lin et al's Concealment Techniques

At ACM CCS'20, Lin et al. [27] introduced eight properties and techniques that can be utilized to conceal HTML elements. They conduct experiments on six popular browsers, testing whether each browser would autofill form elements whose presence is concealed using the respective techniques. Their findings reveal that five of the eight techniques (excluding the CSS *display and visibility* property) are effective against **all** major browsers evaluated. Here, we briefly describe the eight element concealment techniques introduced by Lin et al., focusing specifically on the `<input>` element.

CSS display property. One of the simplest obfuscation approaches involves setting the CSS `display` property of the target element to `none`, which completely removes the element and the space it occupies from the rendered page, as if it never existed. This property also influences the appearance of its child elements.

CSS visibility property. The `visibility` property specifies whether an element is visible, with the `hidden` value causing the element to become imperceptible while reserving its original layout space and position. The `collapse` value exhibits the same behavior for input elements. Like the display property, this property can also cause effects on its child elements. In this work, we regard the *two* properties as separate techniques, as PMs may behave differently when setting `visibility` to different values.

CSS opacity property. By setting the `opacity` property to 0, the element becomes fully transparent and thus invisible to the user, though it maintains its positioned space. Although the `opacity` property is inheritable, the effective opacity of a child element is determined by the product of its own opacity value and that of its ancestors.

Covered by overlay. This deceptive tactic involves positioning a non-transparent overlay element atop the target element, effectively concealing it from the web page view.

Non-effective size. Rendering the element invisible by specifying a non-effective size, achieved by setting its width or height to zero. However, this property has no effect on its child elements by default. Thus, we only consider the property's influence on the `<input>` fields instead of its ancestor elements.

Off-screen placement. Web elements with fixed or absolute positioning can be hidden by displacing them outside the visible device screen area using the top, bottom, left, and right properties.

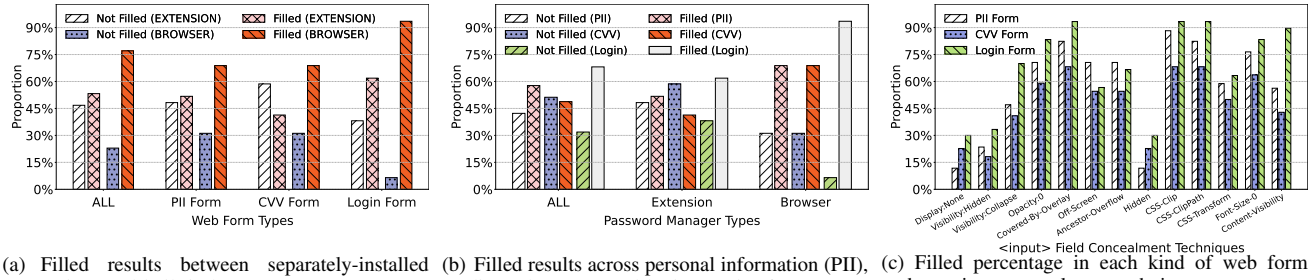
Ancestor's overflow. This approach involves positioning the target element outside the bounds of its ancestor's visible overflow area, making it imperceptible. For example, one could set the ancestor's height/width to zero or adjusting the target's positioning while specifying the `overflow:hidden` property on the ancestor element.

Lin et al.'s work [27] reveals that concealment techniques except `hidden` and `visibility` works in all the browsers. They also present that their considered techniques are most likely not exhaustive and other techniques for hiding `<input>` fields may be feasible.

We consider the above eight techniques introduced by Lin et al. [27], and incorporate seven additional techniques to conduct our experiments. Furthermore, we include the six browsers evaluated by Lin et al. in our experiments to compare the effectiveness of separately-installed PMs and built-in-browser PMs. This also allows us to assess whether major browsers have made improvements in addressing this issue over the past four years.

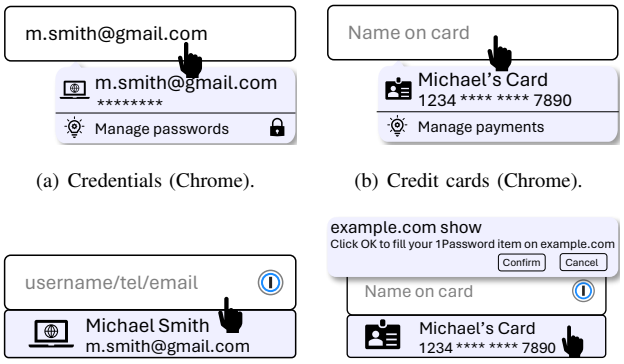
B. Various autofill functionality

Autofill functionality works via detecting web forms and fields, and then determines which data should be filled into forms. For instance, a login form with username and password fields needs to be filled with credentials, while a signup form with name and telephone generally needs to be filled with user information. Besides, different PMs perform variously in triggering the autofill functionality. For instance, Chrome's built-in-browser PM defaults to autofill in `username` and `password` fields on page load (see Figure 6(a)), whereas filling credit card details requires clicking on the respective web form field such as `card-name` field (see Figure 6(b)). Conversely, 1Password [32] requires a click on the `username` field to autofill username and password into the login form (see Figure 6(c)), and displays a confirmation



(a) Filled results between separately-installed (Extension) and built-in-browser (Browser) PMs. (b) Filled results across personal information (PII), credit card (CVV), and login (Login) forms. (c) Filled percentage in each kind of web form under various concealment technique.

Figure 5: Analysis for experimental results when <input> elements are invisible due to their ancestor elements' property.



(a) Credentials (Chrome). (b) Credit cards (Chrome).

(c) Credentials (1Password). (d) Credit cards (1Password).

Figure 6: Autofill functionality of Chrome and 1Password.

prompt before filling in credit card details (see Figure 6(d)). One *common feature* of existing autofill functionality is that, once the autofill functionality is triggered, the PM fills in every detected field in the web form using stored data.

C. Visibility Impact of Ancestor Nodes

We now report the experimental results of autofilling data into <input> fields that are rendered invisible due to their ancestor elements (e.g., elements) being hidden. We conduct these experiments as our case studies in Sec. 4.5 find that PMs may only consider the property of <input> elements. The detailed results are in Table 3 and Figure 5. Non-effective or tiny size of ancestor elements does not influence the width of its child elements unless the value of child elements is set using relative length unit (e.g., em). If we only set the ancestor element's width or height to a non-effective size, the inner <input> element with an absolute length remains visible. Thus, we do not consider these two techniques for ancestor elements. As done in Section 4, we conduct the χ^2 test to explore the influences of PM types, web form types, and concealment techniques on filled results. The hypothesis of each analysis is the same.

The χ^2 test demonstrates that the null hypothesis H_0 in each analysis is rejected and thus the filled probability significantly differs between two PM types (χ^2=42.435, p<0.01), three web forms (χ^2=25.878, p<0.01), and thirteen concealment techniques (χ^2=169.76, p<0.01). Further binary logistic regression test presents that built-in-browser PMs are 2.95 times more likely to fill data into hidden fields than separately-installed PMs. Login forms are the most vulnerable and PMs behave best on credit card forms. Three concealment techniques including `display:none`, `visibility:hidden`, and HTML `hidden` property are properly detected and handled by most PMs.

Comparison with filled probability where <input> fields are invisible due to ancestor elements. We compare the filled probability between cases where <input> fields are directly configured to be invisible from the web page and those are influenced by their ancestor elements. Since we only consider the `non-effective-size` and `tiny-size` properties for <input> fields, we select the remaining 13 concealment techniques for comparison. We use the χ^2 test to evaluate whether the filled probability between the two concealment objects is significantly different. Our hypothesis is as follows: H_0: The filled probability has no significant relationship with how the <input> field is hidden; H_a: The filled probability has a significant relationship with how the <input> field is hidden.

Finally, we fail to reject the null hypothesis (χ^2=0.021, p=0.885>0.05), and accept that the filled probability does not have a significant difference between two concealment objects. Still, four concealment techniques have over five PMs that differ in at least one web form, and 23 PMs have at least one different case (see Tables 2 and 3). Our results indicate most PMs consider the influence of ancestor elements. For `content-visibility`, the filled probability has a significant decrement compared with cases in which <input> fields are configured with this property. We hypothesize that this is because the property is not typically applied to <input> fields directly, and most PMs do not consider this property. However, when applying this property to ancestor elements (e.g., a element), this property functions as intended and is therefore considered by ten PMs, including four chromium-based browsers.

D. Detailed filled probability in our experiments

We have provided the detailed filled probability of different password manager types, web form types, and concealment techniques in Tables 4 and 5. The filled probability is used for drawing our Figures 4 and 5. For instance, for separately-installed PMs in login forms, the filled probability in Figure 4(a) is the number of filled instances (213) divided by all 360 instances in Table-2 (213/360=59.17%).

TABLE 3: Results for our autofill testing experiments, where <input> fields are hidden due to its ancestor elements*.

Form Type: Personal Information

Concealment Tech.	Display: None	Visibility: Hidden	Visibility: Collapse	Opacity: 0	Covered by Overlay	Off-Screen	Ancestor-Overflow	Hidden	CSS Clip	CSS Clip-Path	CSS Transform	Font Size: 0	Content-Visibility
LastPass	×	×	✓	✓	✓	✓	✓	×	✓	✓	✓	×	✓
Avira	-	-	-	-	-	-	-	-	-	-	-	-	-
Norton	-	-	-	-	-	-	-	-	-	-	-	-	-
1Password	×	×	✓	✓	✓	✓	✓	×	✓	✓	×	✓	✓
Bitwarden	×	×	×	×	×	×	×	×	×	×	×	×	×
Kaspersky	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]
Dashlane	×	×	×	×	✓	×	×	×	✓	✓	×	✓	✓
iCloud	-	-	-	-	-	-	-	-	-	-	-	-	-
Keeper	×[2]	×[2]	✓	×[2]	✓	✓	✓	×[2]	✓	✓	×[2]	×[2]	✓
MultiPassword	-	-	-	-	-	-	-	-	-	-	-	-	-
True Key	-	-	-	-	-	-	-	-	-	-	-	-	-
RoboForm	×	×	×	×	×	×	×	×	✓	×	×	✓	×
DualSafe	-	-	-	-	-	-	-	-	-	-	-	-	-
NordPass(Desktop)	×	✓	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	✓
ExpressVPN Keys	-	-	-	-	-	-	-	-	-	-	-	-	-
Dropbox	-	-	-	-	-	-	-	-	-	-	-	-	-
KeePassXC	-	-	-	-	-	-	-	-	-	-	-	-	-
NordPass(Extension)	×	✓	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	✓
Passbolt	-	-	-	-	-	-	-	-	-	-	-	-	-
Proton Pass	-	-	-	-	-	-	-	-	-	-	-	-	-
Microsoft Autofill	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Zoho Vault	-	-	-	-	-	-	-	-	-	-	-	-	-
Enpass	×	×	✓	✓	✓	×	×	×	✓	✓	×	✓	✓
SafeInCloud	-	-	-	-	-	-	-	-	-	-	-	-	-
Chrome	×	×	×	✓	✓	✓	✓	×	✓	✓	✓	✓	×
Edge	×	×	×	✓	✓	✓	✓	×	✓	✓	✓	✓	×
Safari	×	×	×	✓	✓	✓	✓	×	✓	✓	✓	✓	-[1]
Firefox	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Opera	×	×	×	✓	✓	✓	✓	×	✓	✓	✓	✓	×
Brave	×	×	×	✓	✓	✓	✓	×	✓	✓	✓	✓	×

Form Type: Credit Card

Concealment Tech.	Display: None	Visibility: Hidden	Visibility: Collapse	Opacity: 0	Covered by Overlay	Off-Screen	Ancestor-Overflow	Hidden	CSS Clip	CSS Clip-Path	CSS Transform	Font Size: 0	Content-Visibility
LastPass	×[2]	×[2]	✓	✓	✓	✓	✓	×[2]	✓	✓	✓	×[2]	✓
Avira	-	-	-	-	-	-	-	-	-	-	-	-	-
Norton	×	×	✓	✓	✓	×	×	×	✓	✓	×	✓	×
1Password	×[2]	×[2]	✓	✓	✓	✓	✓	×[2]	✓	✓	×[2]	✓	✓
Bitwarden	×	×	×	×	×	×	×	×	×	×	×	×	×
Kaspersky	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]
Dashlane	×	×	×	×	✓	×	×	×	✓	✓	×	✓	✓
iCloud	-	-	-	-	-	-	-	-	-	-	-	-	-
Keeper	×	×	✓	×	✓	✓	✓	×	✓	✓	×	×	✓
MultiPassword	-	-	-	-	-	-	-	-	-	-	-	-	-
True Key	-	-	-	-	-	-	-	-	-	-	-	-	-
RoboForm	×[2]	×[2]	×[2]	×[2]	×[2]	×[2]	×[2]	×[2]	×[2]	×[2]	×[2]	✓	×[2]
DualSafe	-	-	-	-	-	-	-	-	-	-	-	-	-
NordPass(Desktop)	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]
ExpressVPN Keys	✓	×	✓	×	✓	×	×	✓	✓	✓	✓	✓	✓
Dropbox	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
KeePassXC	-	-	-	-	-	-	-	-	-	-	-	-	-
NordPass(Extension)	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]	×[3]
Passbolt	-	-	-	-	-	-	-	-	-	-	-	-	-
Proton Pass	-	-	-	-	-	-	-	-	-	-	-	-	-
Microsoft Autofill	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Zoho Vault	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Enpass	×	×	×	✓	×	×	×	×	×	×	×	×	×
SafeInCloud	×[2]	×[2]	×[2]	×[2]	×[2]	×[2]	×[2]	×[2]	×[2]	×[2]	×[2]	×[2]	×[2]
Chrome	×	×	×	✓	✓	✓	✓	×	✓	✓	✓	✓	×
Edge	×	×	×	✓	✓	✓	✓	×	✓	✓	✓	✓	×
Safari	×[2]	×[2]	×[2]	✓	✓	✓	✓	×[2]	✓	✓	✓	✓	-[1]
Firefox	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Opera	×	×	×	✓	✓	✓	✓	×	✓	✓	✓	✓	×
Brave	×	×	×	✓	✓	✓	✓	×	✓	✓	✓	✓	×

Form Type: Login Information

Concealment Tech.	Display: None	Visibility: Hidden	Visibility: Collapse	Opacity: 0	Covered by Overlay	Off-Screen	Ancestor-Overflow	Hidden	CSS Clip	CSS Clip-Path	CSS Transform	Font Size: 0	Content-Visibility
LastPass	×	×	✓	✓	✓	✓	✓	×	✓	✓	✓	×	✓
Avira	×	×	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	✓
Norton	×	×	✓	✓	✓	×	×	×	✓	✓	×	✓	×
1Password	×	×	✓	✓	✓	✓	✓	×	✓	✓	×	✓	✓
Bitwarden	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Kaspersky	×	×	×	✓	✓	×	✓	×	✓	✓	×	✓	✓
Dashlane	×	×	×	×	✓	×	×	×	✓	✓	×	✓	✓
iCloud	×	×	×	✓	✓	×	×	×	✓	✓	✓	✓	✓
Keeper	×	×	✓	×	✓	✓	✓	×	✓	✓	×	×	✓
MultiPassword	×	×	✓	✓	✓	×	✓	×	✓	✓	✓	✓	✓
True Key	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
RoboForm	×	×	×	×	×	×	×	×	×	×	×	✓	×
DualSafe	×	×	×	×	✓	✓	✓	×	✓	✓	×	✓	✓
NordPass(Desktop)	×	✓	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	✓
ExpressVPN Keys	✓	×	✓	×	✓	×	×	✓	✓	✓	✓	✓	✓
Dropbox	×	×	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	✓
KeePassXC	×[2]	×[2]	×[2]	✓	✓	×[2]	×[2]	×[2]	✓	✓	×[2]	×[2]	✓
NordPass(Extension)	×	✓	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	✓
Passbolt	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Proton Pass	×	×	✓	✓	✓	×	×	×	✓	✓	×	×	✓
Microsoft Autofill	×	×	✓	✓	✓	×	×	×	✓	✓	✓	✓	✓
Zoho Vault	×	×	×	✓	×	×	×	×	×	×	×	×	×
Enpass	×	×	✓	✓	✓	×	×	×	✓	✓	×	✓	✓
SafeInCloud	×	×	×	✓	✓	✓	✓	×	✓	✓	✓	✓	✓
Chrome	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Edge	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Safari	×	×	×	✓	✓	×	✓	×	✓	✓	✓	✓	-[1]
Firefox	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Opera	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Brave	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

* Cell in light purple background means that the filled results is different from cases when the <input> field is directly hidden in Table 2. Other notifications are the same as Table 2.

TABLE 4: Detailed filled probability in our experiments among different password managers and web forms*.

Hidden Object	Input Element				Ancestor Element			
Form Type	**PII**	**CVV**	**Login**	**All**	**PII**	**CVV**	**Login**	**All**
Extension	50.9% (=84/165)	38.8% (=93/240)	59.2% (=213/360)	51.0% (=390/765)	51.7% (=74/143)	41.3% (=86/208)	61.9% (=193/312)	53.2% (=353/663)
Browser	75.3% (=67/89)	75.3% (=67/89)	92.1% (=82/89)	80.9% (=216/267)	68.8% (=53/77)	68.8% (=53/77)	93.5% (=72/77)	77.1% (=178/231)
Summary	59.4% (=151/254)	48.6% (=160/329)	65.7% (=295/449)	58.7% (=606/1032)	57.7% (=127/220)	48.8% (=139/285)	68.1% (=265/389)	59.4% (=531/894)

* Extension for separately-installed password managers; Browser for built-in-browser PMs. PII for personal information forms, CVV for credti card forms, and Login for login forms.

TABLE 5: Detailed filled probability in our experiments among different concealment techniques*.

Hidden Object	Input Element						Ancestor Element					
Form Type	**PII**		**CVV**		**Login**		**PII**		**CVV**		**Login**	
Display: None	11.8%	(=2/17)	18.2%	(=4/22)	26.7%	(=8/30)	11.8%	(=2/17)	22.7%	(=5/22)	30.0%	(=9/30)
Visibility: Hidden	23.5%	(=4/17)	18.2%	(=4/22)	33.3%	(=10/30)	23.5%	(=4/17)	18.2%	(=4/22)	33.3%	(=10/30)
Visibility: Collapse	47.1%	(=8/17)	40.9%	(=9/22)	73.3%	(=22/30)	47.1%	(=8/17)	40.9%	(=9/22)	70.0%	(=21/30)
Opacity: 0	70.6%	(=12/17)	50.0%	(=11/22)	66.7%	(=20/30)	70.6%	(=12/17)	59.1%	(=13/22)	83.3%	(=25/30)
Covered by Overlay	88.2%	(=15/17)	72.7%	(=16/22)	96.7%	(=29/30)	82.4%	(=14/17)	68.2%	(=15/22)	93.3%	(=28/30)
Non-Effectiveness-Size	52.9%	(=9/17)	40.9%	(=9/22)	50.0%	(=15/30)			-			
Off-Screen	58.8%	(=10/17)	45.5%	(=10/22)	50.0%	(=15/30)	70.6%	(=12/17)	54.5%	(=12/22)	56.7%	(=17/30)
Ancestor-Overflow	82.4%	(=14/17)	59.1%	(=13/22)	83.3%	(=25/30)	70.6%	(=12/17)	54.5%	(=12/22)	66.7%	(=20/30)
Hidden	11.8%	(=2/17)	18.2%	(=4/22)	23.3%	(=7/30)	11.8%	(=2/17)	22.7%	(=5/22)	30.0%	(=9/30)
CSS Clip	70.6%	(=12/17)	54.5%	(=12/22)	80.0%	(=24/30)	88.2%	(=15/17)	68.2%	(=15/22)	93.3%	(=28/30)
CSS Clip-Path	88.2%	(=15/17)	72.7%	(=16/22)	100.0%	(=30/30)	82.4%	(=14/17)	68.2%	(=15/22)	93.3%	(=28/30)
CSS Transform	64.7%	(=11/17)	54.5%	(=12/22)	70.0%	(=21/30)	58.8%	(=10/17)	50.0%	(=11/22)	63.3%	(=19/30)
Font Size: 0	70.6%	(=12/17)	59.1%	(=13/22)	73.3%	(=22/30)	76.5%	(=13/17)	63.6%	(=14/22)	83.3%	(=25/30)
Content-Visibility	93.8%	(=15/16)	81.0%	(=17/21)	100.0%	(=29/29)	56.3%	(=9/16)	42.9%	(=9/21)	89.7%	(=26/29)
Tiny Size	58.8%	(=10/17)	45.5%	(=10/22)	60.0%	(=18/30)			-			

* PII for personal information forms, CVV for credti card forms, and Login for login forms. Our work considers 15 concealment techniques for <input> elements and 13 for ancestors.