Evaluating the Security of Password Manager Chrome Extensions

Kevin Geng
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA, USA
khg@andrew.cmu.edu

Marie Moskal
School of Computer Science
Carnegie Mellon University
Pittsburgh PA, USA
mmoskal@andrew.cmu.edu

***Abstract*—Password managers store users' passwords and other sensitive data and are trusted to use this information securely. Developers typically pair password managers with browser extensions to simplify the process for users. However, browser extensions are coded with web APIs, making them vulnerable to some of the same attacks as web applications. They are also susceptible to network attacks that can result in valuable information being leaked. Additionally, password managers have grown in popularity over the past years because of the increased privacy and simplicity that comes with the application. Therefore, we conducted an analysis of the security practices of five Chrome password manager extensions. From the analysis, we identified several practices that should be used by these extensions to improve privacy and security for their users.**

Firstly, manual auto-fill should be the primary form of auto-fill performed by extensions because it limits the attacks that can be executed. The permissions that are needed to achieve automatic auto-fill should only be granted as optionalPermissions and after the user understands the vulnerabilities that could occur by being shown a warning page. Additionally, if automatic auto-fill is being conducted by the extension, the form action must be checked that it is same-origin including the protocol as when the page was first loaded. If same-origin is violated, we suggest that a prompt is displayed to the user before auto-fill is completed to prevent attacks. Finally, we recommend in particular that auto-fill within cross-origin iframes be forbidden entirely except for whitelisted sites, as it violates user expectations and can lead to attacks.

We also recommend that disabling automatic auto-fill when extensions are first downloaded, to help reduce the number of security attacks that extensions are vulnerable to.

***Index Terms*—password managers, browser extensions, browser security**

I. Introduction

Over the past few decades, security experts have encouraged the use of password managers to mitigate the need to remember multiple passwords. Since all accounts are protected with a username and password combination, this means that individuals have to remember all their passwords, which leads them to create simple or reuse the same password for each account. This behavior is very insecure and can lead to attackers gaining access to accounts if they correctly guess a simple password. By using a password manager, users of password managers have to trust extensions to store some of their most valuable information, like passwords to emails and bank accounts. However, since browser extensions are coded with web APIs, they are vulnerable to some of the same attacks as web applications. Due to their importance and widespread use, an analysis of the security practices of these extensions would benefit many users and developers of these extensions.

Browser extensions have introduced serious security threats into browsers and websites that extensions interact with. Google Chrome, in particular introduced a new platform for browser extensions to mitigate several security attacks [1]. However, attacks have discovered novel ways to circumnavigate these protections and continue to attack Chrome extensions. Therefore, in this paper, we evaluate the security practices employed by developers of password manager browser extensions.

There has already been a lot of research conducted on the most popular password managers so we directed our focus to less prevalent password managers that still have a significant number of users. Our security analysis consisted of two parts. Firstly, we conducted a static analysis of each extension by using an existing tool called Tarnish. Secondly, we developed a series of test environments that simulate different scenarios that could result in vulnerabilities enabling an attacker to learn a user's credentials. Some of these situations were also researched and analyzed by Silver et al. [2], but we also looked into circumstances that would result in clickjacking attacks.

Our main contribution is a proposal of several security practices that should be adopted by password managers extensions to improve the security and privacy for users. We suggest that manual auto-fill should be the standard policy used by password managers because it limits the number of security attacks that can be carried out. Additionally, we display a number of attacks that can be carried out by network and web attackers. From our results and observations, it is clear that there are a lot of vulnerabilities in password manager extensions, which means that developers should keep security in mind when creating these tools.

A. Organization

We organize the rest of the paper as follows: Section 2 presents related work that we researched before conducting our own analysis. Then Section 3 provides background on our threat model, password managers, and different types of auto-fill policies. Next, in Section 4, we present the methods we used for analyzing the password manager extensions we

chose. Section 5 presents the results and discussion from our analysis and states the security suggestions we propose. Lastly, in Section 6, we conclude our paper.

II. Related Work

There have been several previous research studies that focused on analyzing password managers extensions and identifying vulnerabilities in the systems. We will identify and summarize them here:

A. Attacks on Password Manager Extensions

1) Clickjacking Attacks: Google Project Zero researcher Tavis Ormandy discovered a bug in the LastPass extension that resulted in a clickjacking attack [3]. This bug worked by convincing users to access a malicious website and then fooling the browser extension to use a password from a previous website on the page, revealing the password to the owner of the malicious site. Ormandy found that a site like Google Translate could disguise the malicious URL and mislead users to visit the malicious website.

2) User Interface Attacks: Carr et al. [4] analyzed five popular password managers by reimplementing disclosed bugs to see if they had been fixed. Additionally, they performed systematic functionality tests, which led them to find four new vulnerabilities in the password managers. They used the vulnerabilities that had already been discovered in certain password managers to determine whether other vendors had similar bugs and security flaws with respect to that vulnerability. In this way, they were able to determine how universal these existing bugs are to other password managers and if there are consistent flaws in commercial password managers that result in security attacks. The new vulnerabilities that Carr et al. discovered were all user interfacing bugs, that were found by testing the password managers under typical operating states. They disclosed these new vulnerabilities they found to the corresponding companies and found that if it was a major high security error, the company quickly fixed it. However, for lower priority problems or issues that don't have an easy fix such as a clipboard vulnerability, the vendors had to make a decision between leaving the bug as is or applying a quick fix that could result in additional vulnerabilities [4].

3) Network Attacker Vulnerabilities: Silver et al. [2] found that auto-fill policies result in many vulnerabilities that are prone to being exploited by network attackers. Through their analysis of browser, mobile, and 3rd party password managers, they found that there are widely different auto-fill policies that each password manager follows. Some of these policies can lead to significant attacks that result in network attackers extracting passwords remotely. One attack they identified was through the use of injecting malicious iframes into a user's webpage, which is particularly easy if the login page is served over HTTP. When the iframes are loaded, the password manager auto-fills the password fields in the iFrame with the user's password information, resulting in the network attacker exfiltrating the user's credentials. In conclusion, Silver et al. proposed that password managers can mitigate most vulnerabilities posed by network attackers by never auto-filling when there are HTTPS certificate validation errors and requiring user interaction in some kind of form [2].

4) Web-based Attacks: In Li et al. [5] research, they focused on analyzing vulnerabilities in web-based password managers. They found that these are susceptible to most of the same vulnerabilities as websites like cross-site scripting (XSS), cross-site request forgery (CSRF), and user interface attacks since most developers fail to understand the security model of the web. Some of the attacks that Li et al. discovered enabled an attacker to obtain complete takeover of the account through an XSS vulnerability or change credentials due to CSRF vulnerabilities. All of these vulnerabilities were discovered through manual analysis, which suggests that further vulnerabilities exist that weren't found by this research group.

B. Defenses

1) Secure filling: Silver et al. [2] proposed a new form of auto-fill called secure filling, which makes auto-filling passwords more secure than manual insertion by users in certain situations. Secure filling is done by the password manager recording the action present in the login form along with the username and password combination. Then when the login form is auto-filled, the password field is made unreadable by JavaScript, which should prevent a network attacker from extracting the password from this field. Additionally, if the username or password fields are modified after auto-fill is complete, either by the user or JavaScript, then the auto-fill aborts and the field is cleared. Lastly, after the auto-fill is submitted and all JavaScript code is run, then the form's action is checked against the stored action from when the password was first saved. If they differ, then the password field is erased and the auto-fill submission fails. Otherwise, the submission completes like normal [2].

C. Methods for Analyzing Browser Extensions

1) Static Analysis: In a study conducted in 2019, Somé [1] assessed the communication APIs used internally by Chrome, Firefox, and Opera browser extensions to communicate between content scripts and background pages. To do this, they developed a static analyzer that detects suspicious communications that extensions are able to hold. The goal was to reduce the number of false positives that the analyzer would make and thereby minimize the need for manual analysis. Through the use of the static analyzer, Somé determined that the privileges that extensions are given can enable web applications to exploit the extension and benefit from the privilege capabilities. Additionally, the privileges, when enabled, give extensions access to user credentials, browsing history, and much more data about users interactions that regular websites don't have. As a result, attackers or other websites that can attack an extension will gain access to these privileges and could use them for malicious deeds [1].

III. Background

A. Threat Model

In our research, we set up our testing environment in the form of a web attacker and network attacker. We assume that the code running in the password manager extension is not malicious and hasn't been compromised. Additionally, in both threat models, we assume that the user is relying on their password manager to manage their credentials. In this section, we will explain the attackers capabilities and goals.

1) Network Attacker: We consider an active man-in-middle network attacker for our threat model. This attacker is able to view and modify network traffic that is being sent or received from a user's machine. For instance, an attacker could be in a local coffee shop and since everyone is on the same WiFi network, the attacker can intercept the traffic. Their goal then is to extract the passwords that are stored in the user's password manager without interacting with the user in any way.

However, we assume that in 2020, all legitimate websites serve their login pages over HTTPS, significantly reducing the capabilities of a network attacker. Only a few years ago, it was common for some websites to serve a login page over HTTP, but submit the credentials over HTTPS, thereby providing confidentiality against a passive network attacker. This approach allows for an active man-in-the-middle attacker to violate integrity by modifying the login page, and thereby intercept data [6]. Thankfully, this state of affairs has changed considerably in the past few years, as browser vendors like Google have strongly pushed for HTTPS adoption by marking HTTP sites as "Not Secure"; as of 2018, 85% of network traffic accessed via Chrome OS is served over HTTPS [7].

However, for sites that do not enable the Strict Transport Security feature, it is still possible for a network attacker to perform an "SSL stripping" attack by redirecting the user to an HTTP version of an HTTPS page, and it is relevant to examine how password managers respond to this situation [8].

2) Web Attacker: The web attacker that we consider is able to control one or more web servers and convince a victim to visit one of its malicious domains. An example of how this can be done is through a phishing attack where the attacker sends the victim an email with the link to one of the malicious domains convincing them to click on the link. Once the link is opened, the site may have a login field that the password manager will fill with the user's credentials if the domain matches one of the saved domains that the password manager has stored. As a result, the web attacker convinces the password manager to provide the credentials for an account because the domain matches but it is actually a fake site controlled by the attacker.

B. Password Managers

Beginning in the early 2000s, companies began developing password managers, which are a storage technology to congregate all usernames and passwords used by an individual for different websites. Since then, many variations have been made and used by millions of individuals. For instance, the LastPass browser extension has over 10 million users [9].

Many security advisors strongly suggest the use of password managers because it helps eliminate the common problem of reusing passwords, which can enable attackers to access many different accounts once they learn the password. Therefore, if the password used for all accounts is leaked due to one website being exploited, it can result in far more danger because of the access that an attacker will gain to other accounts. There are different forms of password managers, ranging from local installed storage or cloud software to web-based services such as browser extensions or token-based hardware devices [10]. In this paper, we will focus on Chrome browser extensions.

Users trust password managers to store all their sensitive information, primarily passwords but they can also hold credit card information. With all this important data in one spot, attackers have been attempting to find ways to exploit and access the passwords. Many security vulnerabilities have been discovered in password managers, especially with browser extensions. However, security professionals still advise that individuals continue to use password managers because they are critical in keeping people from reusing passwords, despite the security flaws that have surfaced in many of the top password managers [11].

C. Types of Auto-fill

Auto-fill is the mechanism that is used by password manager extensions to populate login fields that are displayed on a webpage, which eliminates the need for individuals to remember every password that they have set. Password manager extensions use several auto-fill policies, which can lead to vulnerabilities such as a network attacker extracting passwords from the user's password manager without any interaction with the user. From the research conducted by Silver et al. [2], they determined that all the extensions they analyzed had auto-fill policies that were too loose resulting in vulnerabilities that could expose user's credentials to attackers. This is due in part to password managers injecting custom fields into the DOM, which could potentially be tampered with by attackers. We will now discuss different types of auto-fill policies and special situations that could warrant different behavior by the extension to prevent security attacks.

1) Manual Auto-fill: The user needs to interact with browser extension or web browser in some way before the extension is able to complete auto-fill. The types of interaction that qualify are clicking on or typing in the username field, pressing a keyboard shortcut, or pressing a button in the browser or extension window [2].

2) Automatic Auto-fill: Once the login page is loaded, the extension can auto-fill the login fields with the user's credentials. No form of interaction is needed by the user, which makes this a preferred method by many individuals because it quickens the process for accessing accounts. However, this method also introduces many unexpected consequences that can result in security vulnerabilities.

Many password managers employ a mix of these two types of auto-fill in an attempt to make the experience more pleasant for users but also to mitigate the vulnerabilities that occur

when automatic auto-fill is employed. Some situations that would cause a change in the policy from automatic to manual auto-fill are when the username and password fields are in an iframe, policies that differ from when the credentials were initially stored or auto-filling invisible fields. It was found that LastPass allows auto-filling within iframes for usability purposes, but this also leads to clickjacking attacks [12].

IV. Methods

Our analysis was to conduct a survey of five different Chrome browser extensions for password management. We focused on considering a network attacker and web attacker as our threat model. In this section we will cover the different extensions we picked and how, the static analysis we completed of the extensions, and the testing we completed in different environments by setting up fake domains and different situations of HTTPS.

A. Browser Extensions

We picked 5 browser extensions from the Chrome extension store after trying several of the many different password managers that are available. Many research papers have analyzed the most popular password managers such as KeePass and 1Password. Therefore, we decided to focus our analysis on password managers that were less common but still had a significant number of users. We picked ones that had between 100,000 and 400,000 users and had been updated within the past two months. In addition, we included LastPass in our analysis because it was one of extensions that fascinated us the most when exploring the different password managers. The following are the 5 browser extensions we used for our analysis:

- LastPass
- Bitwarden
- Keeper
- Enpass
- Blur

B. Static Analysis

One of the methods we used to analyze the password managers was conducting a manual analysis of the extensions. To do this, we used the CRX browser extension to obtain all the source code for each extension. We also found a static analysis tool online, Tarnish, which identifies possible vulnerabilities in the extension [13]. It helped us to identify the permissions and content script policies set by the extension. It also listed potentially dangerous functions that are used in the extension, along with which file and line they are used and files that could lead to clickjacking attacks.

Most of the extension code was minified, which made it very difficult to create a comprehensive method to complete manual analysis. Although the Chrome Web Store prohibits uploading extensions that have obfuscated code [14], it is still acceptable to minify code, which involves removing variable names, restructuring the control flow, and possibly merging the application code with library code, such as jQuery. Although

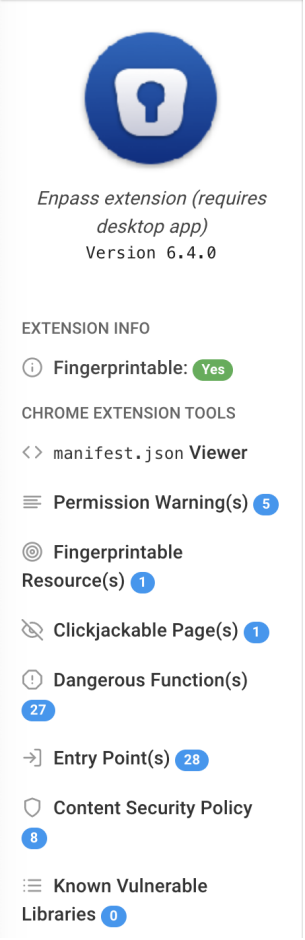

Fig. 1. Information that the Tarnish static analysis tool provides about a Chrome extension.

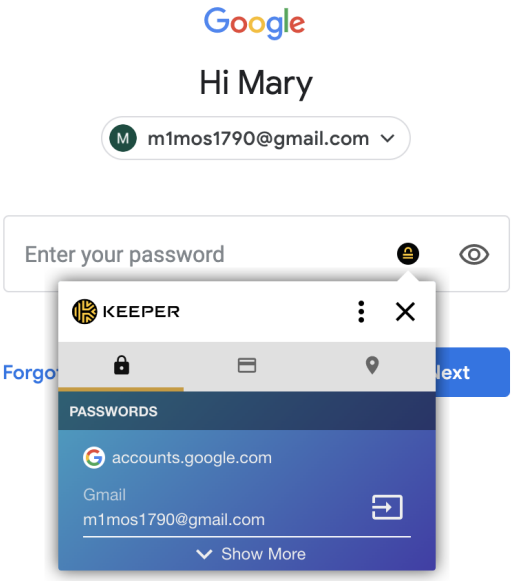

Fig. 2. Example of the Keeper Extension in use

it is possible to "beautify" JavaScript code to make the whitespace more readable, it can be challenging to recover the author's original intention. As a result, we decided to use a static analysis tool to find vulnerabilities in the password managers. D. F. Somé [1] used static analysis methods to detect if there was a possibility that extensions were exposing vulnerable APIs to web origins. Though we investigated the use of these tools, the static analysis tool was relatively rudimentary and we decided to search for other similar tools that would help us complete static analysis. Additionally, we looked into developing our own testing environments to further the analysis, which we will discuss next.

C. Testing in Different Environments

In order to test the behavior of password managers in different login scenarios, we created a series of small web pages that would be able to test these behaviors. These pages are simple and succinct, with the goal of being the minimal possible test cases that would be able to expose different behaviors, which avoids having to deal with the complexity of actual pages.

The test pages were set up on two different domains, `bank.local` and `evil.local`. These domains were hosted locally by running an Nginx instance inside a Docker container that served both sites from the same IP address, and by modifying the hosts file so that those two domains would resolve to the Nginx instance. Additionally, in order to properly serve those sites over HTTPS, a fake TLS certificate was generated so that the Nginx instance could use it to serve both domains.

For testing purposes, we used the latest stable version of Google Chrome at the time of writing, Chrome 81. We installed each of the tested extensions into a separate installation of Google Chrome, and enabled only one extension at a time in order to prevent conflicts between extensions. Furthermore, we installed the root authority for our fake TLS certificate into this installation, so that the `bank.local` and `evil.local` domains would show as trusted.

The different login pages are listed as follows:

Same-origin auto-fill policies

1) HTTPS origin → HTTPS action
2) HTTP origin → HTTPS action
3) HTTP origin → HTTP action
4) HTTPS origin → HTTP action
5) Different action on load
6) Different action on submit
7) Form with autocomplete off

iframe auto-fill policies

1) Same-origin iframe
2) Cross-origin iframe
3) Both same-origin and cross-origin iframe

V. Results

A. Permissions for Auto-fill

Table I shows the permissions enabled for each of the password managers we analyzed. Each extension has the `http://*/*` and `https://*/*` permissions or `<all_urls>` permissions set. These permissions are of particular concern because they allow the extension to read and modify all the data on all websites that the user visits, which includes the DOM of all the sites [13]. http://*/* and https://*/* provide the extension with the same capabilities as the `<all_urls>` permission so we will only reference the `<all_urls>` from now on, but are referring to `http://*/*` and `https://*/*` as well. With this permission, the extension has more capability than it requires to perform auto-fill functionality. In order to auto-fill a username and password field, the extension needs to be able to modify the DOM of the login page. This is typically the current page that the user has shown on their desktop. Therefore, the extension doesn't need to access each page open in a user's browser and its data, only the current page the user is viewing, which leads us to conclude that these permissions are too relaxed.

The `activeTab` permission gives the extension temporary access to the current tab that the user is on once the user has invoked the extension. It requires the user to click on the browser action or context menu for the extension. Once the user navigates to a different site or closes the tab, the access is revoked [15]. This permission is an alternative to the `<all_urls>` permission, which we have seen all of the password managers use instead. When an extension only needs to perform its functionality after being invoked or requested to by the user, `activeTab` permission is only needed to complete this. As stated on the activeTab webpage, if the `activeTab` permission isn't used, then the extension will need to have "full, persistent access to every web site" with the `<all_urls>` permission [15]. In the case of password managers, to complete manual auto-fill, which requires interaction from the user before the password field is filled, the `activeTab` permission provides sufficient privilege to the extension to complete this task. The user already has to interact with the webpage before auto-fill is completed, and through this interaction, it will invoke the extension to gain privileges to the current tab that has the login fields displayed. However, if a password manager wants to use the automatic auto-fill policy, the `activeTab` permission will not give the password manager the adequate capabilities to do this.

We recommend that password manager extensions only need the `activeTab` permission to complete the capabilities desired by their users. Additionally, if the extension were to be compromised by an attacker, the attacker would only obtain access to the current tab that the user is viewing once the user has explicitly invoked the extension. In contrast, with the `<all_urls>` extension, the attacker would obtain access to all the data that the extension can access without any regulation. Therefore, the best security practice would be for

TABLE I
PERMISSIONS ENABLED FOR EACH PASSWORD MANAGER EXTENSION

Bitwarden	Blur	Enpass	Keeper	Lastpass
tabs clipboardRead http://*/* https://*/* file:///*	tabs <all_urls> http://*/* https://*/*	<all_urls> tabs webNavigation http://*/* https://*/*	tabs <all_urls>	tabs webNavigation http://*/* https://*/* chrome://favicon/* file:///* https://lastpass.com

password managers to only use the `activeTab` permission and not the `<all_urls>` permission.

A downside to this practice would be that password managers are only able to implement manual auto-fill. As a result, automatic auto-fill wouldn't be a feasible policy anymore. In the face of security, this is a positive consequence because as Silver et al. [2] found, most automatic auto-fill password managers they analyzed are vulnerable to attacks they discovered, such as redirect sweep and password exfiltration attacks. However, from the user perspective, most individuals prefer automatic auto-fill because it is far more convenient and doesn't require any interaction for the password manager to immediately fill in the login fields once the page is loaded. To help user experience but also improve security practices in password managers, we propose that automatic auto-fill is an optional feature, that is only enabled once the user views a through warning page and agrees to the possible vulnerabilities that could occur as a result of using automatic auto-fill.

In order to do this in a secure manner, extensions should use the optionalPermissions key in `manifest.json` and add `<all_urls>` to this field, which is the same syntactically as the permissions field . At runtime, the chrome.Permissions API will request access to the optionalPermissions by showing a prompt to users that explains why the permissions are needed and to only grant permissions that are necessary [16]. Since most password manager users aren't security focused individuals, this warning prompt should be designed to give the most clarity about the issues and vulnerabilities that could arise once these permissions are granted without going too in depth about the specific attacks that could happen. By using optionalPermissions for the `<all_urls>` field, the extension is more secure since it is only running permissions that are enabled and needed and the user is more informed about which permissions are running when the automatic auto-fill feature is enabled [16]. As a result, password managers will be able to implement the automatic auto-fill feature once users decide to enable it and give access to the permissions, which leads to better security and a more sophisticated user.

B. Auto-fill Policies

1) Transport security: For login pages 1-4, we wanted to test what happens when a login page is served over HTTP, and/or it submits to a form over HTTP. Either of these cases have the potential to cause a user's password to be leaked. If the login page is served over HTTP but submits to an HTTPS

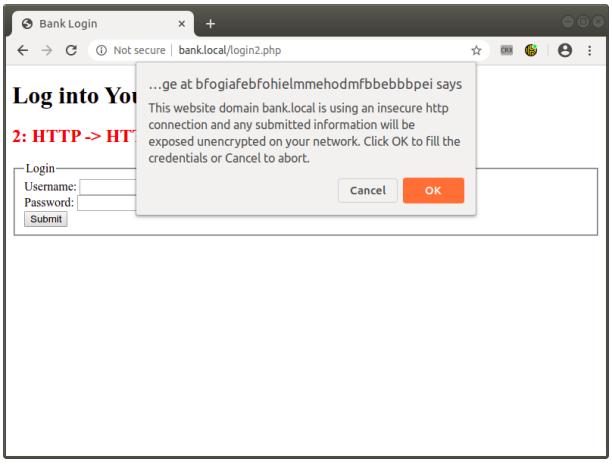

Fig. 3. The Keeper extension displays a warning when trying to submit a login form served over HTTP. The warning is triggered on manual auto-fill, or automatic auto-fill when enabled.

form, an active network attacker can inject JavaScript into the login page and steal the password as the user types it in. On the other hand, if it submits to a form over HTTP, then the credentials will be sent unencrypted, and a passive network attacker will be able to read them.

When it comes to password managers, however, the relevance of these situations is that if a user originally saves their password over an HTTPS connection, then is a network attacker able to exfiltrate their password at a later point in time? As previously mentioned, for domains that do not have Strict Transport Security enabled, an active network attacker would be able to perform an "SSL stripping" attack to redirect the user to a fake login page served over unencrypted HTTP. If the password manager auto-fills the user's password into the site, then the passwords from a large number of sites can be stolen using a sweep attack, as proposed by Silver et al [2]. On the other hand, almost all websites these days are sensible enough to avoid submitting login credentials over an insecure connection, but we still wanted to test whether extensions would attempt to detect this situation and respond appropriately.

This scenario is not as uncommon as it might appear; for instance, an attacker could set up a fake Wi-Fi network offering free internet access in a public location. Indeed, in certain situations, users are already accustomed to ignoring security warnings, for instance when clicking through interstitials to

TABLE II
AUTO-FILL POLICIES: TRANSPORT SECURITY

	Default (HTTPS)	HTTP	Broken HTTPS	HTTP form action	Autocomplete "off"
Bitwarden	Manual	(no change)	(no change)	(no change)	(no change)
Keeper	Manual[a]	+ Warning	(no change)	(no change)	(no change)
Blur	Automatic	Manual	(no change)	Manual	(no change)
Enpass	Manual	+ Warning	(no change)	(no change)	(no change)
Lastpass	Automatic	Manual[b]	(no change)	(no change)	(no change)
Chrome	Automatic	No fill	No fill	(no change)	(no change)

[a]Allows opting in to automatic fill.
[b]Automatic if the password was originally saved over HTTP.

log in to a Wi-Fi network. For this reason, we also tested password manager behavior when a site was served with a broken HTTPS connection. Although the user may not actually intend to log into a site with an insecure HTTPS connection, if a password manager auto-ills their password into that site, then the password will already have been compromised.

The results from our experiments are described in Table II. We find that although some extensions we tested warn about HTTP connections, unfortunately not all do. Furthermore, only the Chrome browser itself is able to detect the presence of broken HTTPS. Even years later, the policies of extensions in this area have not evolved significantly from those discovered by Silver et al. [2], and we hope they will do so.

2) Form actions: It is possible for the target of a form action to be on a different origin than the original site. Silver et al. [2] discussed the effects of changing a form's target action. Since most password managers don't actually check the target of a form being submitted, and since this data is only visible to the DOM and not to the user, it's impossible for users to know where their credentials are actually being sent.

However, it is unlikely that this attack could be taken advantage of given our threat model, which assumes that login pages are served over HTTPS; as such, changing the form action would require the ability to serve arbitrary content, which is not the case on most websites. One possible attack scenario would be embedding a form into an HTML email with a password field, but webmail clients have long realized the risks of doing so, and show warnings when such forms are submitted [17].

As shown in Figure 4, we found that Blur appears to be able to detect that the form action is a different origin, since it switches to manual auto-fill. However, it does not warn the user of what is happening. In contrast, LastPass disables manual auto-fill while also warning the user that their credentials are being submitted to a different origin.

However, another vulnerability that Silver et al. described [2] is that the form action could be changed immediately before submitting the form. This case is not detected by any of the extensions we tested. In fact, it would be likely be very difficult to do so — doing so would likely require using the `webRequest` API, which would break certain login forms using AJAX, and significantly increase complexity. Furthermore, this attack would only be realistic if an attacker already had script execution on the page, in which case the password would already be compromised. As such, we more or less reject the importance of this behavior as a flaw in password managers.

TABLE III
AUTO-FILL POLICIES: FORM ACTIONS

	Default (HTTPS)	Cross-origin form action on load	Cross-origin form action on submit
Bitwarden	Manual	—	—
Keeper	Manual[a]	—	—
Blur	Automatic	Manual	—
Enpass	Manual	—	—
Lastpass	Automatic	Manual + Warning	—
Chrome	Automatic	—	—

[a]Allows opting in to automatic fill.

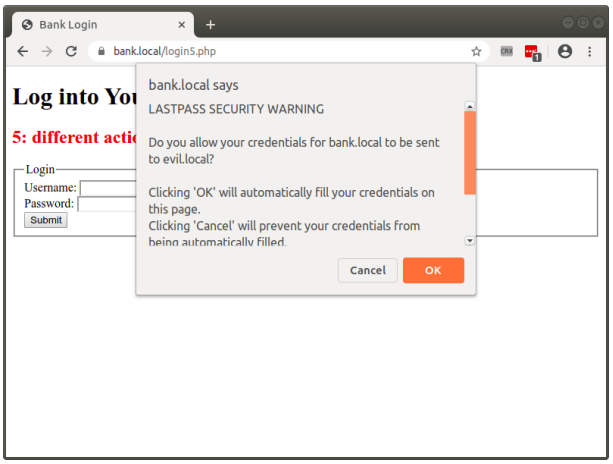

Fig. 4. The LastPass extension displays a warning when the login form action is submitting to a different origin. (However, a form action of HTTP is treated as same-origin, and no warnings are emitted.)

3) Auto-fill in iframes: An additional area of concern is performing auto-fill within cross-origin iframes. Our results are described in Table IV. As Silver et al. [2] observed, password manager extensions that auto-fill passwords into iframes can be abused to steal passwords quickly from a large number of origin sites, without the user being aware. However, in the scenarios we considered, this would require another vulnerability to already be present, in order to modify the

TABLE IV
AUTO-FILL POLICIES: IFRAMES

	Default (HTTPS)	Same-origin iframe	Cross-origin iframe
Bitwarden	Manual	—	Manual for top-level frame
Keeper	Manual[a]	—	No fill[b]
Blur	Automatic	—	Automatic for content frame
Enpass	Manual	—	Manual for content frame
Lastpass	Automatic	—	No fill + Warning[b]
Chrome	Automatic	—	Manual for content frame

[a]Allows opting in to automatic fill.
[b]Shows UI corresponding to fill for content frame, but fill does not work.

iframed content: for example, if a network attacker was able to serve HTTP content on a certain domain, or if a web attacker was able to find an XSS vulnerability in the website.

A similar concern is that a web attacker could use this to enable a CSRF attack through clickjacking. For example, some sites like GitHub require the user to enter their password before performing sensitive operations, such as transferring the ownership of a repository. However, this would also require an existing CSRF vulnerability, and would require the target site to allow such pages to be iframed, which would be unlikely. As such, performing auto-fill within cross-origin iframes should not pose a direct threat to security on its own.

Nonetheless, we recommend that auto-fill be disabled within cross-origin iframes, an approach that the LastPass extension takes, as it violates user expectations — the fact that an iframe is cross-origin is not apparent to users in any way, so users will likely not expect to be performing auto-fill for a different domain than the one mentioned in the address bar. Some sites may require this behavior for backwards compatibility, but it would be better to add such sites to a whitelist to prevent this behavior from being more broadly taken advantage of.

However, out of the extensions we tested, we did observe a concerning behavior from the Bitwarden extension, as shown in Figure 5. Although this extension only performs auto-fill on manual request, when it does so, it fills in all forms on the page with the saved credentials of the top-level frame. This includes forms inside cross-origin iframes, even if a form is already present for the current origin. This would allow a cross-origin iframe to steal the user's passwords for the top-level frame. In fact, this issue was discovered by a security audit of Bitwarden in 2018 [18]. However, the security issue was not fixed due to the assumption that the presence of a malicious iframe implies that "the website (or device) is already in a compromised state". We do not necessarily agree that this is the case: for instance, websites embed advertisements in iframes from different origins, but use sandboxing techniques to ensure safety. In this scenario, passwords could be stolen even if JavaScript is disabled in the iframe, for example by using the CSS exfiltration technique [19].

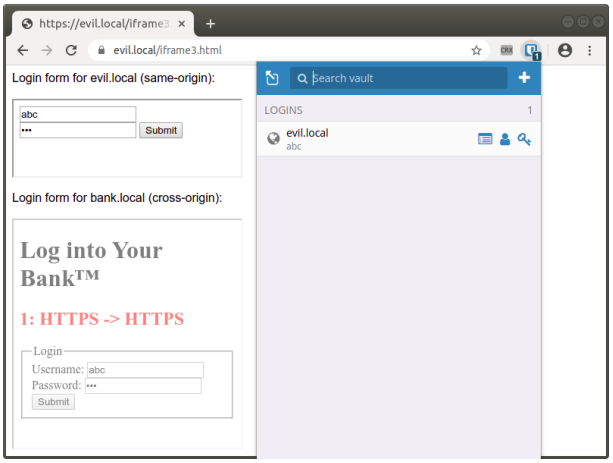

Fig. 5. An example showing Bitwarden's behavior on a website containing both same-origin and cross-origin iframes. The credentials for the source origin are filled into both frames.

TABLE V
WEB ATTACKS ON EXTENSIONS

	Login form in the DOM	Shows auto-fill UI in DOM
Bitwarden	No (browser action)	No
Keeper	No (browser action)	Yes[a]
Blur	Yes (iframed)	On form selection[a]
Enpass	No (native app)	No
Lastpass	No (separate tab)	Yes[a]
Chrome	No (browser UI)	No (browser UI)

[a]Requires clicking in iframe first.

C. Web Attacks on Extensions

Finally, we analyzed the susceptibility of various extensions to web attacks themselves, as shown in V. As previously mentioned, although extensions use a different JavaScript heap, they sometimes choose to inject elements into websites using content scripts, which users can interact with.

As a result, content injected by an extension is usually placed inside of an iframe so that the DOM is separate as well, so that the target page cannot directly control the extension. One example of dangerous extension UI to inject, however, is that used for logging into the extension itself. This is because if users become habituated to this, the login form could be impersonated by a malicious website, which would then be able to steal the password manager's password, similar to a phishing attack. We discovered that only the Blur extension exhibited this property.

Another possible vector for abuse is showing other UI in the target DOM that can be subject to a clickjacking attack. We successfully performed such an attack on the LastPass extension, convincing the user to auto-fill their password without their knowledge, thereby bypassing the theoretical protection offered by requiring users to manually select a UI element to auto-fill their password.

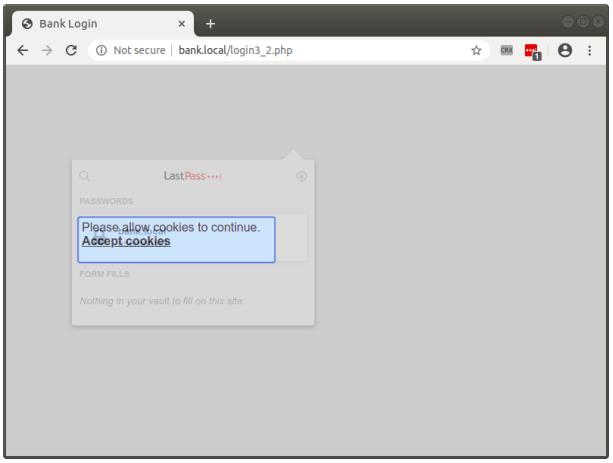

Fig. 6. We performed a clickjacking attack on the LastPass extension's injected UI, causing the user to enter their password without their knowledge. We reduced the opacity to make the LastPass UI is visible, but it is possible to make it completely invisible.

VI. Conclusion

In our project, we studied several different aspects of the security of password managers, focusing specifically on five Chrome extensions. For each of these extensions, we analyzed specific behaviors and permissions, and comprehensively tested the effects of different auto-fill situations.

Many of the investigations we performed, and the security recommendations we made, began with the paper by Silver et al. [2]. Although that paper was published six years ago, it is unfortunate to see that in some ways, the policies of various extensions have not changed. For instance, none of the extensions we tested is able to detect broken HTTPS connections, which could compromise user security.

More and more frequently, password managers are being recommended to members of the general public as a way of securely managing passwords, since it is difficult to remember more than a few passwords. However, this means that these password managers have a special responsibility to protect the passwords that they store. For instance, the average user may not know the difference between HTTPS and HTTP, so it falls on the password manager to prevent the user from submitting their password over an insecure channel, and/or adequately inform them of the risks. If this is not done, they risk reducing rather than increasing security for average users.

We hope that the existence of projects like ours will cause developers to further consider the policies used in their extensions, and how they can make them secure by default, for all users.

Acknowledgment

We would like to thank the professor of our Browser Security class, Hanan Hibishi, and our TAs, Wai Tuck Wong and Pragati Mishra, for their support and feedback throughout this project.

Source Code

We have placed all code for our tests in a GitHub repository at https://github.com/gengkev/14828-final-project.

References

[1] D. F. Some, "Empoweb: Empowering web applications with browser extensions," *2019 IEEE Symposium on Security and Privacy (SP)*, 2019.
[2] D. Silver, S. Jana, D. Boneh, E. Chen, and C. Jackson, "Password managers: Attacks and defenses," Aug 2014. [Online]. Available: https://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-silver.pdf
[3] J. Porter, "Lastpass fixes bug that could let malicious websites extract your last used password," Sep 2019. [Online]. Available: https://www.theverge.com/2019/9/16/20868111/lastpass-bug-exploit-password-manager-malicious-website
[4] M. Carr and S. F. Shahandashti, "Revisiting security vulnerabilities in commercial password managers," 2020. [Online]. Available: https://arxiv.org/pdf/2003.01985.pdf
[5] Z. Li, W. He, D. Akhawa, and D. Song, "The emperors new password manager: Security analysis of web-based password managers," Jul 2014.
[6] T. Hunt, "Your login form posts to https, but you blew it when you loaded it over http," May 2013. [Online]. Available: https://www.troyhunt.com/your-login-form-posts-to-https-but-you/
[7] E. Schechter, "A milestone for chrome security: marking http as "not secure"," Jul 2018. [Online]. Available: https://www.blog.google/products/chrome/milestone-chrome-security-marking-http-not-secure/
[8] J. Ali, "Performing & preventing ssl stripping: A plain-english primer," Aug 2018. [Online]. Available: https://blog.cloudflare.com/performing-preventing-ssl-stripping-a-plain-english-primer/
[9] "Lastpass: Free password manager." [Online]. Available: https://chrome.google.com/webstore/detail/lastpass-free-password-ma/hdokiejnpimakedhajhdlcegeplioahd?hl=en
[10] "Password manager," May 2020. [Online]. Available: https://en.wikipedia.org/wiki/Password_manager
[11] G. Fowler, "Review — password managers have a security flaw. but you should still use one." Feb 2019. [Online]. Available: https://www.washingtonpost.com/technology/2019/02/19/password-managers-have-security-flaw-you-should-still-use-one/
[12] [Online]. Available: https://bugs.chromium.org/p/project-zero/issues/detail?id=1930
[13] "Please submit an extension above to begin." [Online]. Available: https://thehackerblog.com/tarnish/
[14] "Trustworthy chrome extensions, by default," Oct 2018. [Online]. Available: https://blog.chromium.org/2018/10/trustworthy-chrome-extensions-by-default.html
[15] "The activetab permission." [Online]. Available: https://developer.chrome.com/extensions/activeTab
[16] "chrome.permissions." [Online]. Available: https://developer.chrome.com/extensions/permissions
[17] C. Coyier and C. Coyier, "Html forms in html emails: Css-tricks," Dec 2011. [Online]. Available: https://css-tricks.com/html-forms-in-html-emails/
[18] .-B. S. LLC, "Bitwarden security assessment report," Nov 2018. [Online]. Available: https://cdn.bitwarden.net/misc/Bitwarden%20Security%20Assessment%20Report%20-%20v2.pdf
[19] "Stealing data with css: Attack and defense." [Online]. Available: https://www.mike-gualtieri.com/posts/stealing-data-with-css-attack-and-defense