Fluid Passwords - Mitigating the effects of password leaks at the user level

Michael Farcasin  
Oklahoma State University  
Stillwater, Oklahoma  
Email: farcasi@cs.okstate.edu

Akhileshwar Guli  
Oklahoma State University  
Stillwater, Oklahoma  
Email: akhileg@ostatemail.okstate.edu

Eric Chan-Tin  
Oklahoma State University  
Stillwater, Oklahoma  
Email: chantin@cs.okstate.edu

Abstract—Password leaks have been frequently reported in recent years, with big companies like Sony, Amazon, LinkedIn, and Walmart falling victim to breaches involving the release of customer information. Even though passwords are usually stored in a salted hash, attackers still guess passwords because of insecure password choices and password reuse. However, the adverse effects of a password breach can be mitigated by changing users’ passwords. We introduce a simple yet powerful algorithm to reset user account passwords automatically, while still allowing users to authenticate without any additional effort on their part. We implemented our algorithm as a Firefox add-on that automatically resets a user’s password when they log in to their account, and stores the new password in the built-in Firefox password manager.

I. INTRODUCTION

Breaches that leak millions of passwords, e.g. LinkedIn [23], [13], [15], Yahoo [21], [12], Walmart [17], and iCloud [22] are becoming a daily news event. But while password leaks can lead to both financial loss and loss of privacy [9], [10], organizations can be slow to detect breaches [31].

Many methods proposed to protect users from the effects of password leaks, such as OAuth [29], SAuth [16], and Two Factor Authentication [24], require server-side implementation. Despite these solutions, users still have to create strong, unique, memorable passwords and reset their passwords often. But the amount of effort required to create and maintain such passwords leads users to create similar, sometimes the same, weak passwords for multiple accounts, and make only minor changes when required to change their passwords [3].

We propose an effort-free, client-side solution to the problem of password theft in the form of a Firefox add-on to automate password resets, which we call Fluid Passwords. The idea behind our add-on is: for a given website, after a user logs in, seek out the site’s password reset page, generate a secure random password, and update the password for the account. By leveraging Firefox’s built-in password manager, we can then store the updated password for use at the next login, all without requiring any interaction from the user. By leveraging Firefox Sync, users can access both their stored passwords and the add-on on any computer they login with, effectively reducing the number of passwords any one person has to remember to the one they use for Firefox Sync. But the Fluid Passwords add-on is not itself a password manager - instead, its primary task is to automatically find a password reset page and reset a password so the user doesn’t have to.

We demonstrated the add-on’s success on 23 out of 29 websites from Alexa’s top 100 websites [1], and provide suggestions for making it work on the remaining 6 sites in section V-D. Fluid Passwords addresses 3 main problems:

1) Creating a strong and memorable password is difficult.
2) Creating a unique password for every site is difficult.
3) Changing passwords is difficult, and especially changing passwords often enough to foil attackers who gain access to an old password is difficult.

Although solutions to the problems exist, client-side solutions to changing passwords are still imperfect and require user interaction citegottLastpass, dashlanePasswordChanger. In the case of users on a single computer, i.e. users who don’t need Firefox Sync, our add-on solves all of these problems. In the case of users on multiple computers, our add-on reduces the problems to the problem of maintaining a single password, i.e. the one to access Firefox Sync. We claim this add-on helps mitigate the effect of password leaks by continuously changing a user’s passwords.

Our main contributions are:
- Creating an algorithm to refresh user passwords that doesn’t require user interaction.
- Creating an algorithm to find a website’s password-reset page.

We also wrote scripts to automatically submit password-reset forms and login forms. We implemented these in a Firefox add-on, evaluated its performance, and gave it a comparative rating as an authentication scheme.

The rest of the document is organized as follows. In section II we review previous literature. In section III we detail our algorithm’s design and implementation. In section IV we detail the algorithm’s successes and failures. In section V we examine the usability and security of the implementation. In section V-D we discuss potential improvements. We conclude in section VI.

II. BACKGROUND

A. Protection After a Leak

Past research has focused on helping users create strong passwords, e.g. with PwdHash [23], an add-on that turns a
simple password into a long and random one, allowing users to create memorable passwords and have good security. In the case of a breach where password hashes are stolen, strong enough passwords resist even attacks by experts [27]. However, password-strengthening solutions like PwdHash cannot, by design, protect a user once their password is stolen or guessed. In that scenario, a user must reset his or her password to prevent an attacker from using it, and must change it enough that the attacker cannot guess the new one.

Previous work to protect user accounts in the face of a password breach has focused mainly on server-side solutions. O-Auth [29] is a partial solution that provides authenticated parties with only the privileges those users should be allowed. However, there are studies that show that OAuth is intrinsically vulnerable to App Impersonation attack [11].

SAuth [16] employs authentication synergy among different web services. A user wishing to access service S must also authenticate their account on service V, where S and V are any two web services that are used daily. For example, if a user wants to login into Facebook the protocol redirects user to also authenticate their account on Google. However, unless the user may use the same password for both services S and V, and if not, then it places an additional burden on users as they have to remember two passwords to access a single service, and it does not prevent attackers from accessing all the services available to a legitimate user.

Two-factor authentication (2FA) [24] is an increasingly-popular protocol in which users authenticate with two different forms of authentication. For example, users might have to enter a temporary password/confirmation code provided via SMS or email, or authenticate with a smart card [30]. OPass [25] is one such protocol that adds an additional one time key, sent to the user’s cell phone at the time of authentication. Another 2FA protocol uses ambient noise to authenticate without user interaction [14]. As the authentication requires a separate authentication token in addition to a password, users are safe even in case of a password leak. But because of the inconvenience of waiting for the one-time password, confirmation code, or ambient noise tests, these methods are not commonly used.

B. Protection from Password Leaks

In addition to securely storing passwords with salted hashes, various algorithms and techniques have been developed to detect password leaks and cracking [5], [19]. But these solutions are often complex and costly to deploy and maintain.

Hardware scrambling is another way to avoid password leaks [6]. Trusted hardware such as Scramble S-Crib [7] is used to scramble passwords before they are hashed on the server. The server hardware holds the encryption keys that scramble the password, so this hardware is required to do any password guessing. Hence even in case of hash leaks, attackers cannot crack the passwords.

Many websites and organizations follow the practice of password expiration on an interval, e.g. 1 to 3 months, after which the users have to change their existing passwords. However, this approach is unscalable for users [26]. As a workaround for this problem, users use similar or the same passwords for multiple accounts and across password changes [8].

In contrast to the above solutions, our add-on works on the client-side, so any user can employ it for themselves, it works on multiple sites, and it is effort-free for the user. It is free, and requires no special hardware or reliance on third-party software. The only restriction is that users use the Firefox web browser.

III. DESIGN & IMPLEMENTATION

A. General Overview

The flow diagram for Fluid Passwords is illustrated in figure 1. The steps are as follows:

1) Identify when the user accesses a login page. When the user submits the login form, go to the next step.
2) Capture the login credentials. After a successful login, go to the next step.
3) Search for a link to the site’s password reset page.

Fig. 1. A high-level description of the algorithm.
a) First, check for the site in a file that maps a site’s base address to its password-reset page (which we called purls.txt). If the site is found, go to the next step.
b) Otherwise, search the current page for the URL most likely to lead to the password reset page.

4) Push the current tab onto a stack and open the selected URL in a new tab.
   a) If the new page is a password reset page, generate a new random password and submit the password reset request. Then store the new password in Firefox’s password manager. If this site wasn’t in purls.txt, store its site/password-reset-page URL pair now.
   b) If the new page is a login page, enter the captured credentials, submit the login form, and return to the previous step.
   c) Otherwise return to the previous step.

5) Close any extra tabs opened by the program.

We chose a successful user-login as the start time for our add-on for two reasons. First, it provided a simple way to obtain the user’s credentials in case they weren’t already stored in Firefox’s password manager. Second, it means the add-on runs whenever the user logs in, which we expect to be far more often than the 60-day password expiration guideline published by NIST [20].

However, it is important to note that if the add-on cannot find the password reset page, then no password is changed; so the add-on fails safely. In addition, all the websites we observed had only one password reset page, so the add-on will not accidentally change the password saved in Firefox’s password manager without actually resetting the password on the site.

The critical steps are as follows:

1) **Identifying the Login Page**
   - Login pages can almost always be identified by searching for a form with a pattern of text field, password field and a submit button.

2) **Capture Login credentials**
   - After identifying the fields above, we simply get the text in those fields.

3) **Identifying Successful Login**
   - Most of websites redirect to the login page on a failed login attempt; therefore a successful login can be identified by failing to find a login form after entering the credentials.
   - We can also identify failed attempts by searching for strings that indicate an error, like “invalid password”, or “incorrect username”.

4) **Selecting a URL**
   - Each URL is assigned a priority value. Priority values are chosen based on the text string that an `<a>` tag encapsulates, with higher priority values given to the strings that are more likely to lead to a password reset page. Then the URL with the highest priority value is chosen. For example, a hyperlink with the text “password” has higher priority value than a hyperlink with the text “account”, so the algorithm would select the URL in the tag with the first string [1]. The current implementation has 10 priority levels, as shown in table I. We were unable to find any literature on the subject, so values were chosen based on our own observations of the websites listed in table II. These were the most common links used to navigate for password reset. For example, on Facebook we navigate from Settings → General → Edit Password. The further away a link is from the password reset page, the lower priority value it received.
   - The URL with highest priority value on the current page is selected. Then the current tab is pushed onto a stack, and the selected URL is opened in a new tab. We repeat this process until we land on password reset page.
   - If the new tab does not produce any URL, we move back to the previous tab (popping the current tab from the stack) and select the URL with next highest priority.

5) **Opening the URL in a New Tab**
   - The add-on opens the new tab in the background. While users can see the tabs that the add-on opens and closes, the tab the user logged in with remains the active tab (assuming the website doesn’t change it), so the work in the background tabs does not affect the user’s experience.

6) **Identifying a Password Reset Page**
   - As we were unable to find any literature on identifying password reset pages, we designed our algorithm based on our own observations. Password reset pages have various designs, but most of them have a form with a text field (for usernames), three password fields, and a submit button. We can

| Hyper link pattern | Priority level |
|--------------------|----------------|
| privacy            | 0              |
| setting            | 1              |
| profile            | 2              |
| account            | 3              |
| security           | 4              |
| preference         | 5              |
| my login           | 6              |
| edit profile       | 7              |
| password           | 8              |
| change password    | 100            |

**TABLE I**

PriOrity anD PatterNs

1At this time the algorithm has hardcoded English phrases it searches for, though in the future it would be appropriate to include search terms in an easily-updated file, especially one with multiple languages to account for non-English websites.
identify a password reset page by searching for a form with this design.

B. Password Generation

The contribution of Fluid Passwords is the automated password reset after successful logins and finding the password reset page for each website. Any password generation algorithm can be used to generate a secure password to be stored in the browser’s password manager. We created a simple password generator for the add-on to make it compatible with the websites we tested on, and examine the security of that generator in this section.

The password generator produces pseudorandom 12-character strings from the 95 printable ASCII characters minus the space character, because we found the space character did not work on many websites. In other words, the set of 94 characters [0-9A-Za-z`~!@#$%ˆ&∗()_+=−[]\{}—;':",./<>?].

It uses Stanford JavaScript Crypto Library \[18\] as a secure pseudorandom number generator to generate passwords. We chose to create 12-character passwords so the add-on would work with sites that require a maximum password length of 12 characters. Sites with stricter requirements are discussed in section \[V-D\].

Since passwords are chosen probabilistically, we can measure the security of the passwords with Shannon entropy. Low entropy would imply the password distribution is fairly predictable. For example, the entropy of the English alphabet is between 0.6 to 1.3 bits per character. Formula \[1\] shows that we need a minimum 6.55 bits per character to encode the passwords in binary form. Since we cannot use only part of a bit, we require a minimum 7 bits per character, meaning a 12-character password requires 84 bits to encode.

\[
H(X) = -\sum_{i} P(x_i)\log_2 P(x_i) = -\sum_{i=1,...,94} \frac{1}{94}\log_2 \left(\frac{1}{94}\right) = 6.5546...
\]

RFC 4086 recommends 56 bits for higher-than-moderate-security passwords in 2015 \[3\]. Our algorithm far exceeds that, but it falls short of the recommended 88 bits for a very-high security password \[4\]. We could improve the security by adding non-ASCII characters or lengthening the password. However, using non-ASCII characters would make retyping passwords difficult for those users who wished to do so, and lengthening the passwords would cause some websites to reject them for being too long. Therefore, we chose to use the current algorithm at this time.

C. Implementation

Since web browsers are the most common interfaces for accessing online accounts, we chose to implement our algorithm as a Firefox add-on.

For testing, we used a computer with an Intel(R) Core(TM) i7-4510U processor, 8GB RAM. It had a 64-bit Windows 8.1 Operating System, and we developed our add-on with cfx for FireFox version 42.0, using Mozilla Firefox Addon-SDK 1.17 and JQuery 1.11.3.
IV. Evaluation

A. Websites

We tested our add-on with sites chosen from Alexa’s top 100 websites in December 2015 [1]. A detailed list of all the websites is given in appendix A. Table II lists the websites we chose to test our add-on with, and whether or not the add-on supported these websites. In total, we tested our add-on on 29 of the top 100 sites.

Table II lists the websites we considered. In the case of websites where the add-on didn’t work, we give the limitation. Limitations included: being unable to fetch a reset link from a button, and being unable to trigger keyboard event listeners for form inputs when we updated the text in an input box. We address these limitations in section V-D. We note that when the add-on doesn’t work, it has no adverse effects on a user’s experience, and the websites that the user visits still work. On these sites, the user would have to change their password manually, as if they didn’t have the add-on.

We list the 72 websites that we did not choose to test on, along with the reasons, in appendix A. Although Google was the most popular site on the list, we chose not to consider it because changing a Google account password causes Android tablet- and smartphone-users to login again with their new password. We ignored websites in a foreign language, since the add-on can only match hyperlink strings against text patterns in English. We ignored websites that ask for security questions as part of a password reset after logging in, as it requires more user-interaction to automate such actions. We ignored torrent and adult websites on ethical grounds, and we ignored websites with no logins, such as news sites, or websites which served as landing pages for other sites, since they do not require users to login.

Table III lists the percent CPU usage and memory usage (in MB) by Firefox web browser during the process of password reset, with and without the add-on enabled. The first two columns are when the add-on was enabled, and the third and fourth columns are when the add-on was not enabled. The last two columns show the difference in overhead caused by the add-on. Overall, the difference is negligible, with the add-on adding only 1.48% more CPU usage and 16.54 MB more memory usage on average. And it is worth noting that these times were tested the first time we accessed a site. Because the add-on stores the password reset URL and in future logins can directly open the password reset page instead of searching again, we expect it to use even less overhead during future logins.

The add-on’s listeners take a few seconds to attach to the web page elements, and they must wait until the web page loads all its content. When the add-on’s listeners fail to attach, or the user submits the form before the listeners are attached, the add-on does not continue executing: it neither opens any tabs in the background nor resets the user’s password.

B. Improvements & Accuracy

In this section, we discuss the various improvements we made as we went through different versions, making the add-on support websites with different designs. Depending upon the basic flow from a login page to a password reset page, websites can be broadly classified into three categories.

1) Simple Websites: These websites have a simple design where the landing page/home page has a basic login form with a user name, a password and a submit button. The design of the password reset page is also simple with an optional username, three password fields, and a submit button. Figures 2 and 3 show the design for simple login pages. Facebook has a login page on its landing page, whereas Yahoo redirects to a new page with simple login design. Figure 4 shows a basic password reset page from Linkedin. The control flow for this website, identifying the login page to submitting the password reset form, is straightforward. 30.7% of the websites in table II fall under this category. Our initial implementation worked only with these types of websites.

2) Websites that require re-login: These websites ask a user to login again before they can change their password. 19.3% of the websites in table II fall under this category, including Amazon and eBay. To support these websites, we added an additional check for a new login page and added code to wait on the redirect to password reset page.

3) Complex Websites: We chose priority values for URLs based on our observation that the URL would lead to password reset page. Therefore, it’s not always certain that we will...
find the password reset page. The initial algorithm design would stop when both the current page was not a password reset page, and the search process found no new links. But e-commerce sites such as Amazon and eBay have a large number of hyperlinks, and in fact 50% of the websites listed in Table II fall under this category. To handle these scenarios, we implemented a depth-first search, in which failure causes the algorithm to go back to the previous tab and select the URL with the next-highest priority. In theory, this design would exhaustively search a site’s URLs and eventually find the password reset page. However, we set a threshold of 20 opened pages, after which the algorithm terminates. Figure 6 shows how this might work in practice.
1) Select URL-3 (the highest priority URL) and open it in a new tab.
2) The new tab with URL-3 has 3 URLs. URL-5 has the highest priority level, so open URL-5 in a new tab.
3) The new tab has no new URLs, so we go back to the previous page.
4) URL-7 has the next highest priority, so we open it in new tab.
5) URL-9 has the highest priority in this tab, and the process continues.

C. Limitations

The add-on does not support websites that send a confirmation code or reset link to user’s email/phone, or that ask for security questions in order to reset a password. And many banking websites require users to enter a CAPTCHA to login, so our add-on does not support those websites when a re-login is required.

In addition, the current implementation does not support any Google-based website, because Google’s password reset page leverages a div tag for form submission.

On the other hand, Wordpress and Netflix both listen for actual keyboard events before allowing form submission, so simply assigning a value to a text box with textBox.value does not work in this case.

We provide suggestions for addressing these limitations in section V-D.

V. DISCUSSION

A. Comparative Rating

We rate our add-on using Bonneau’s framework for comparative evaluation of authentication schemes [4]. Naturally, our scheme inherits many of its qualities from passwords and Firefox’s password manager, though in some cases it exceeds both.

Usability

1) It is memorywise-effortless for users at a single computer, and quasi-memorywise-effortless for users of Firefox Sync, as the sync password is the only password users need to remember.
2) It is scalable-for-users, as the add-on and password manager handle all the passwords.
3) It is nothing-to-carry.
4) It is quasi-physically-effortless, for the same reasons Firefox’s password manager is.
5) It is easy-to-learn, because passwords are.
6) It is efficient-to-use, more so than passwords because this does not require typing.
7) The current version does not have infrequent-errors. The ideal version (that works on every website) would have infrequent-errors, as users need only click the login button.
8) It does not have easy-recovery-from-loss, for the same reasons Firefox’s password manager doesn’t.

Deployability

1) It is accessible, for the same reasons Firefox’s password manager is.
2) It has negligible-cost-per-user.
3) The current version is quasi-server-compatible, as it works on some websites but not others. The ideal version would be server-compatible.
4) It is not browser-compatible, though in future work we would extend the add-on to other browsers.
5) It is not mature, as it is newly-created.
6) It is non-proprietary.

Security

1) It is quasi-resilient-to-physical-observation, like Firefox’s password manager.
2) Likewise it is quasi-resilient-to-targeted-impersonation.
3) It is resilient-to-throttled-guessing, as it generates secure, random passwords.
4) It is resilient-to-unthrottled-guessing, as there are $94^{12} \approx 2^{78}$ possible passwords.
5) It is not resilient-to-internal-observation.
6) We claim it is quasi-resilient-to-leaks-from-other-verifiers, both because it generates unique passwords for each site and because it resets passwords relatively quickly.
7) It is resilient-to-phishing.
8) It is resilient-to-theft under the same assumptions that Firefox’s password manager is.
9) It does not offer no-trusted-third-party.
10) It requires-explicit-consent.
11) It is unlinkable, because it generates random passwords.

B. Firefox Sync

Firefox Sync is a service offered by Firefox wherein users can create an account on Firefox that allows them to share information across devices, especially passwords and add-ons [2]. Users login to their account with a username and password, so Firefox Sync acts as an online password manager with a master password.

When users have Fluid Passwords without using Firefox Sync, they are limited to storing their passwords on one device, so that Fluid Passwords will not function on other devices (because it won’t be able to access the user’s login credentials). However, when users employ both Fluid Passwords and Firefox Sync, and share passwords and add-ons across devices, Fluid Passwords will function any time the user is using Firefox, and their updated passwords will be stored and accessible on any of their devices running Firefox.

C. Other considerations

The add-on is very simple to use, as it only requires the user to run Firefox and install the add-on. However, the current version of the add-on only works on websites for which the user has stored his or her credentials in Firefox’s password manager. For example, if a user stores their Facebook credentials in the password manager, but not their LinkedIn credentials, then the add-on would only work with their Facebook account, not LinkedIn. Similarly, the add-on doesn’t work for multiple users. For example, if two users, A and B, both use the add-on, but A stores their Facebook password in the password manager and B doesn’t, then the add-on would only change A’s password. And Fluid Passwords may cause problems for users who use the add-on on one device, e.g. their computer, but also need to login to a site with another device that doesn’t have access to the add-on, e.g. an iPhone. With the current version of the add-on, the only way for such a user to login with their second device would be to look up their password in Firefox’s password manager, or on Firefox Sync if they use it. However, because Fluid Passwords relies on Firefox’s password manager, the cost of losing a password is as low as using the password manager alone. The user would need to reset their password and store the new password in their password manager.

Once the add-on has successfully reset a password on a site, it stores the URL of the password reset page in a separate file named purls.txt. After that, whenever it encounters the same website again, it uses the link in purls.txt to go directly to password reset page. This has 3 main implications:

1) Users can share purls.txt files to increase our pool of url mappings. In particular, users could submit their purls.txt files to a central server, which could then update a master list and distribute it to users, effectively crowdsourcing password-reset-URL searching.
2) Sites that have personalized URLs for their password-reset pages cannot be usefully stored in purls.txt, and the add-on will be forced to search for the URL each time the user logs in.
3) It allows the add-on to work on non-English sites. Any site in the purls.txt file ignores the (currently) English-only url-searching algorithm and jumps directly to the password reset page, where it looks only at form element types, which are all written in HTML language.

We must also discuss the insecurity of an add-on that automatically fills in user credentials when it finds a login form. Once the user triggers the add-on’s automatic password reset functions by logging in, it may navigate to sites with malicious scripts that request login credentials to steal them. We don’t have a good solution for this problem, except to rely on the sites themselves to keep the pages a user sees after logging in free of such scripts.

D. Future Work

Capturing login credentials is difficult for websites where the web page has dynamic forms, forms nested in iframes, and other complexities.$^5$ Identifying a successful login becomes difficult when we have hidden or dummy login-forms, or when it is difficult to identify a login page.$^6$

Traversing URLs is the next crucial task to reach password reset page. Our current version looks for text hyperlinks on the page to assign priority, but this method fails when the URLs are embedded as hyperlinks in images or buttons. In addition, identifying a password reset page is difficult when forms on the page do not follow the normal pattern.$^7$ Finding a single algorithm for all sites on the web would be an important step forward.

Two sites stand out in particular: Google and Dropbox. Google’s password reset page has a div tag with a click-listener for its form submission. Assuming the tag has the same id for every user, we might attempt to find the tag and issue it a JavaScript click. Dropbox listens for keyboard events to fill in its password reset form, denying our script the ability to do so, and the tags in its forms have enough unique information to prevent our simply creating our own input tags with the required information. We might try simulating keyboard events with JavaScript to fill in such forms.

While our primary concern is getting our add-on to work on every site on the web, we would ideally also extend our

$^5$ e.g. zillow.com, etsy.com, buzzfeed.com
$^6$ e.g. godaddy.com has a dummy login form on the user’s profile page
$^7$ e.g. aliexpress.com, alibaba.com, flipkart.com
$^8$ Optional username, three password fields, then a submit button. E.g. wordpress.com, themeforest.net
add-on to Google Chrome, Safari, and other browsers, as well as mobile phones. The add-on currently does not mesh well with mobile phone usage, since mobile phones typically do not have the same add-ons installed as desktop computers. However, Firefox and Chrome at least allow mobile phone users to sync their passwords across devices, so would have up-to-date passwords on their mobile phones.

We would also like to add the ability for users to blacklist sites that they do not want the add-on to function on. This would allow users to handle passwords for those sites on their own, while still gaining the benefit of the add-on on other sites. And such a blacklist could be seeded with sites that require passwords shorter than 12 characters, so that the add-on doesn’t waste time attempting to create a new password for that site every time a user logs in. We note, however, that the add-on does not break on any site - it simply exits without changing a user’s password and closes any open tabs, keeping the process invisible to the user.

The code for generating passwords currently only produces random 12-character passwords, but some sites have more exact requirements, e.g. at least 1 uppercase, 1 lowercase, 1 number, and 1 symbol. Future work for the add-on would include detecting a site’s password requirements and modifying the password generation algorithm to create passwords that fit them.

Last, since the goal of Fluid Passwords is to provide better usability and security, a user study would confirm our tool is user-friendly and non-intrusive, but we expect that to be the case due to the design of the algorithm. We stress that this add-on works in the background, so it does not interfere with a user’s browsing experience.

VI. Conclusion

We developed an algorithm to automatically reset user passwords and implemented it in Fluid Passwords, a Firefox add-on that leverages Firefox's password manager to improve the security and usability of passwords online. Fluid Passwords solves the problem of password reuse, and reduces the problems low password strength, and the difficulty in making frequent password changes to a single password, allowing users to put more focus on maintaining a secure password while expending less effort overall. In its current version, it works on 23 out of 29 sites from Alexa’s top 100 sites, and has inherent functionality to work on many more, and adds negligible cost to performance with only a small cost to login-time.

While there are many more improvements that can be made, this is currently the only client-side solution we know of to keep accounts secure after a password leak without user interaction. We plan to release Fluid Passwords as an official Firefox add-on.

VI. Acknowledgments

Thanks to the contributors of Mozilla’s Addon-SDK tutorials and API, and thanks to Dr.-Ing. Roland Bless for creating a bibtex citation of RFC 4086.

References

[1] Top 100 websites. [http://www.alexa.com/topsites] 2015. [Online; accessed Feb, 2015].
[2] Mozilla firefox web browser — keep your firefox in sync. [https://www.mozilla.org/en-US/firefox/sync] 2017. [Online; accessed 13-February-2017].
[3] D. E. 3rd, J. Schiller, and S. Crocker. Randomness Requirements for Security. RFC 4086 (Best Current Practice), June 2005.
[4] J. Bonneau, C. Herley, P. C. Van Oorschot, and F. Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In Security and Privacy (SP), 2012 IEEE Symposium on, pages 553–567. IEEE, 2012.
[5] N. Chakraborty and S. Mondal. A new storage optimized honeyword generation approach for enhancing security and usability. CoRR, abs/1509.06094, 2015.
[6] S. crib. Scramble SCRib - No More Password Leaks. [http://www.newswire.com/scramble-scrib-no-more-password/251627] 2013. [Online; accessed 28-Nov-2013].
[7] D. Cvrcek. Hardware Scrambling No More Password Leaks. [https://www.lightbluetouchpaper.org/2014/03/07/hardware-scrambling-no-more-password-leaks/] 2014. [Online; accessed 7-March-2012].
[8] M. Farcașin and E. Chan-tin. Why we hate it: two surveys on pre-generated and expiring passwords in an academic setting. Security and Communication Networks, 2015.
[9] E. Grosse. Gmail account security in Iran, 2011. [Online; accessed 13-February-2017].
[10] M. Honan. How Apple and Amazon security flaws led to my epic hacking. [http://www.wired.com/2012/08/apple-amazon-mat-honan-hacking/] 2012. [Online; accessed 13-February-2017].
[11] P. Hu, R. Yang, Y. Li, and W. C. Lau. Application impersonation: problems of oauth and api design in online social networks. In Proceedings of the second edition of the ACM conference on Online social networks, pages 271–278. ACM, 2014.
[12] C. Hubbell. Huge Yahoo Password Leak — 453,441 Passwords Exposed. [https://geeks.onlinenlarge-huge-yahoo-password-leak-453441-passwords-exposed/] 2012. [Online; accessed 12-June-2012].
[13] M. JIMENEZ. Over Six Million Encrypted LinkedIn Passwords Leaked Online. [http://www.prweb.com/releases/prweb2012-6/prweb9582548.htm] 2012. [Online; accessed 07-June-2012].
[14] N. Karapanos, C. Marforio, C. Soriante, and S. Capkun. Sound-proof: Usable two-factor authentication based on ambient sound. arXiv preprint arXiv:1503.03700, 2015.
[15] S. M. Kelly. Linkedin Confirms, Apologizes for Stolen Password Breach. [http://mashable.com/2012/09/06/linkedin-passwords-hacked-confirmation/#DnD4MWQwItuq3] 2012. [Online; accessed 06-June-2012].
[16] G. Kontaxis, E. Athanasopoulos, G. Portokalidis, and A. D. Keromytis. Sauth: Protecting user accounts from password database leaks. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, pages 187–198. ACM, 2013.
[17] M. Kumar. Hackers leak 13,000 Passwords of Amazon, Walmart and Brazzers Users. [http://thehackernews.com/2014/12/password-hacking-data-breach.html] 2014. [Online; accessed 27-December-2014].
[18] S. C. S. Lab. Stanford javascript crypto library. [http://bitwiseshiftleft.github.io/sjcl/]
[19] H. Mohammad, C. N. Gutierrez, M. J. Atallah, and E. H. Spafford. Ersatzpasswords: Ending password cracking and detecting password leakage.
[20] NIST. Usenix security repository/checklist/download?id=898&cid=1. 2015.
[21] A. A. POPKIN. 2 million stolen passwords for Facebook, Twitter, Google, Yahoo and others leaked online. [http://www.nbcnews.com/technology/2-million-stolen-passwords-facebook-twitter-google-yahoo-others-leaked-2011691030] 2013. [Online; accessed 04-December-2013].
[22] L. Razavi. The iCloud leak: weak security isn’t only a problem for Apple’s backup service. [http://www.newstatesman.com/sci-tech/2014/09/icloud-leak-weak-security-ist-only-problem-apples-backup-service] 2014. [Online; accessed 02-September-2014].
[23] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. C. Mitchell. Stronger password authentication using browser extensions. In Usenix security, pages 17–32. Baltimore, MD, USA, 2005.
APPENDIX

1) Google.com
2) Instagram.com
3) Google.es
4) Indeed
5) Facebook.com
6) Msn.com
7) Googleadservices.com
8) Cnn.com
9) Youtube.com
10) Microsoft.com
11) Netflix.com
12) Amazon.in
13) Baidu.com
14) Ahrefs.com
15) Amazon.de
16) Go.com
17) Yahoo.com
18) Amazon.co.jp
19) Stackoverflow.com
20) Google.co.id
21) Amazon.com
22) Google.co.uk
23) 360.cn
24) Xinhuanet.com
25) Wikipedia.org
26) Reddit.com
27) Craigslist.org
28) Blogger.com
29) Qq.com
30) Ask.com
31) Tianya.cn
32) Google.com.au
33) Twitter.com
34) Google.fr
35) Diply.com
36) nytimes.com
37) Google.co.in
38) Google.com.br
39) Ok.ru
40) Bbc.co.uk
41) Taobao.com
42) Tmall.com
43) Google.ca
44) People.com.cn
45) Live.com
46) Oneclickads.net
47) Alibaba.com
48) Cntv.cn
49) Sina.com.cn
50) Pinterest.com
51) Google.com.mx
52) Pixnet.net
53) Linkedin.com
54) Wordpress.com
55) Pornhub.com
56) Gmw.cn
57) Yahoo.co.jp
58) Paypal.com
59) Google.com.hk
60) Ebay.de
61) Weibo.com
62) Mail.ru
63) Naver.com
64) Google.pl
65) Ebay.com
66) Tumblr.com
67) Amazon.co.uk
68) Googleusercontentcontent.com
69) Google.co.jp
70) Imgur.com
71) Ups
72) Dailymotion.com
73) Yandex.ru
74) Sohu.com
75) Xhamster.com
76) Google.co.kr
77) Blogspot.com
78) Xvideos.com
79) Rakuten.co.jp
80) Wikia.com
81) Vk.com
82) Google.ru
83) flickr.com
84) Chindia.com.cn
85) Hao123.com
86) IMDb.com
87) Kat.cr
88) Dropbox.com
89) Tco
90) Apple.com
91) Github.com
92) Livedoor.jp
93) Bing.com
94) Google.it
95) Soso.com
96) Ebay.co.uk
97) Google.de
98) Fc2.com
99) Nicovideo.jp
100) Dailymail.co.uk

[24] B. Selvarajan. Simple two-factor authentication, May 10 2007. US Patent App. 11/267,148.
[25] H.-M. Sun, Y.-H. Chen, and Y.-H. Lin. opass: A user authentication protocol resistant to password stealing and password reuse attacks. Information Forensics and Security, IEEE Transactions on, 7(2):651–663, 2012.
[26] B. Ur, F. Noma, J. Bees, S. M. Segreti, R. Shay, L. Bauer, N. Christin, and L. F. Cranor. "I added ‘!’ at the end to make it secure": Observing password creation in the lab. In Proc. SOUPS, 2015.
[27] B. Ur, S. M. Segreti, L. Bauer, N. Christin, L. F. Cranor, S. Komanduri, D. Kurilova, M. L. Mazurek, W. Melicher, and R. Shay. Measuring real-world accuracies and biases in modeling password guessability. In 24th USENIX Security Symposium (USENIX Security 15), pages 463–481, 2015.
[28] Wikipedia. 2012 linkedin hack — wikipedia, the free encyclopedia, 2015. [Online; accessed 14-November-2015].
[29] F. Yang and S. Manoharan. A security analysis of the oauth protocol. In Communications, Computers and Signal Processing (PACRIM), 2013 IEEE Pacific Rim Conference on, pages 271–276. IEEE, 2013.
[30] G. Yang, D. S. Wong, H. Wang, and X. Deng. Two-factor mutual authentication based on smart cards and passwords. Journal of Computer and System Sciences, 74(7):1160–1172, 2008.
[31] V. J. Yodaiken. Systems and methods for detecting a security breach in a computer system, Sept. 8 2009. US Patent 7,587,763.
| Reason                                    | Websites (count: 71) |
|------------------------------------------|---------------------|
| App breaks the live smartphone connection| Google.com(1)       |
| No Login(2)                              | Ask.com(33), Cnn.com(76) |
| Torrent(1)                               | Kat.cr(71)          |
| Adult(3)                                 | Xvideos.com(45), Pornhub.com(63), Xhamster.com(68) |
| Landing page for other sites(2)          | Go.com(78), Onclickads.net(37) |
| Security Question(7)                     | Live.com(12), Bing.com(24), Msn.com(27), Microsoft.com(28), Aliexpress.com(29), Apple.com(48), Alibaba.com(61) |
| Duplicate(12)                            | Youtube.com(3), Google.co.in(10), Blogspot.com(20), Google.co.uk(31), Googleleadservices.com(51), Google.ca(60), Blogger.com(81), Google.com.au(83), Googleusercontent.com(92), Amazon.co.uk(66), Amazon.in(77), Ebay.co.uk(99) |
| Foreign website(44)                      | Baidu.com(4), Qq.com(8), Taobao.com(11), Sina.com.cn(13), Yahoo.com.jp(15), Weibo.com(16), Google.co.jp(18), Yandex.ru(19), Hao123.com(22), T.co(23), Google.de(25), Amazon.co.jp(30), Google.fr(34), Google.com.br(35), Tmall.com(36), Mail.ru(41), Sohu.com(44), Google.ru(46), Google.it(49), Google.es(50), Amazon.de(53), 360.cn(55), Tianya.cn(57), Diply.com(58), Ok.ru(59), Google.com.mx(62), Google.com.hk(64), Naver.com(65), Rakuten.co.jp(69), Soso.com(73), Nicovideo.jp(74), Google.co.id(79), Xinhuanet.com(80), Bbc.co.uk(84), Pople.com.cn(85), Cntv.cn(86), Pixnet.net(87), Gmw.cn(88), Ebay.de(89), Google.pl(91), Google.co.kr(94), Chinadaily.com.cn(96), Livdoor.jp(98), Dailymail.co.uk(100) |