FLUIDPASSWORDS - MITIGATING THE EFFECTS OF PASSWORD LEAK AT USER LEVEL

By

Akhileshwar Guli

Bachelor of Technology
Information Technology
Jawarharlal Nehru Technological University
Hyderabad, Andhra Pradesh
India
2011

Submitted to the Faculty of the
Graduate College of
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 2015

FLUIDPASSWORDS - MITIGATING THE EFFECTS OF PASSWORD LEAK AT USER LEVEL

Thesis Approved:

Dr. Eric Chan-Tin

Thesis Advisor

Dr. K. M. George

Committee Member

Dr. Nohpill Park

Committee Member

Acknowledgments

This Thesis is dedicated to my parents late Mr. Kasinath Guli, Mrs.Chandrakala Guli and my family members for supporting me through out my life.I would like to thank my friends who helped me out with their moral support when I needed the most. Without their support I would not be able to complete my thesis.

I would also like to thank my advisor, Dr. Eric-Chan Tin, for believing in me,for your encouragement and support, my committee members, Dr. Nohpill Park, for helping me develop creative and logical thinking and Dr. K. M. George, for the support you have shown through out my Masters. Without your guidance, patience and time, I would not be where I am today. I admire you all, and I hope that you continue to put your full efforts into teaching and research.

Thank you Michael Farcasin for constant help, advice and support during the development, for teaching me what you know about Javascript and Add-on development. You are a wonderful friend and it was a pleasure working with you.

Lastly, I would also like to thank all of the faculty and members of Computer Science Department. You all gave the opportunity to strengthen my technical and communication skills.

Thanks to Dr.-Ing. Roland Bless for creating a bibtex citation of RFC 4086.

Name: AKHILESHWAR GULI

Date of Degree: DECEMBER, 2015

Title of Study: FLUIDPASSWORDS - MITIGATING THE EFFECTS OF PASSWORD LEAK AT USER LEVEL

Major Field: COMPUTER SCIENCE

Abstract:

Password leaks have been frequently reported in the recent years, with many big companies such as Sony, Amazon, LinkedIn, and Walmart falling victim to breaches involving the release of customer information. Even though passwords are usually stored securely in a salted hash, the availability of powerful password cracking platforms have still enabled attackers to crack passwords. And with the increase in on-line infrastructure, attackers have an easier time than ever breaking into business databases. Therefore, we need better password protection at the user level, and ideally one that does not cost users any additional effort. We know that the adverse effects of a user's password being guessed can be mitigated by changing the user's password. Therefore, 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 in a Firefox Add-on that automatically resets a users password when they log in to their account, and stores the new password in password manager.

Contents

List of Tables

Table Page

List of Figures

Figure Page

Chapter 1

Introduction

Most web services have authentication systems to prevent attackers from impersonating a legitimate user, and password-based authentication is the most widely-used of these systems. Although long and complex passwords are hard to guess, there are no precautionary measures in place to protect accounts in case of a password leak. Since password leaks can lead to both financial loss and loss of privacy [1] [2], and because organizations can be slow to detect breaches[1] [3], we need a system that protects passwords in advance of a password leak.

Some of the ways of mitigating the effects of a password leak are:

I. **Strong Passwords**: A straight forward and simple way is to use strong random passwords, in which case attackers may not be able to guess a password even if they are stolen. Using a third party password generator or using pre-generated random passwords by few websites would be a better option for using strong passwords. However, most users do not prefer these methods as they

[1]Security programs monitor the entire application periodically, for the attack to happen without being noticed it has to happen within the time window

are hard to remember and learn a new one when changed [4].

II. **Authentication Methods and Protocols**: Since password based authentication method is inadequate and vulnerable to attacks various other authentication methods involving biometrics had been developed. These type of authentication systems are mostly used at offices or restricted areas, but not for on-line accounts. Various authentication protocols have been suggested such as:

- OAuth [5] for OSNs, i.e. providing users with a separate and more secure page for logging in than they use to browse the site normally, and then giving those sessions only the privileges those users should be allowed. One example of the benefits of OAuth is the breach of a Twitter app called Tweetgif: the information for thousands of Twitter accounts were leaked, but no Twitter credentials were compromised because Tweetgif used OAuth. However, there are studies that show that OAuth is intrinsically vulnerable to App Impersonation attack [6].
- SAuth [7] which employees authentication synergy among different web services, in which case user willing to access on service S will also have to authenticate for their account on service V, where S and V can be any regular web services that are used daily. For example, if a user wants to login into Facebook the protocol redirects user to also authenticate his account on Google. Access to Facebook is allowed only if the user has successfully authenticated on Google and Facebook. Hence in case of

password leaks of one services the attackers cannot access the information of that service as they will need the password of the second service to authenticate. Although this ensures good security it would not be effective if the password for both the services is the same and as it adds an extra step in the authentication process, it is an additional burden on users as they have to now remember two passwords.

- Two Factor Authentication (2FA) [8], this is a two-step process similar to SAuth, with few variations in the second step such as entering a temporary password/confirmation code (provided to user via SMS or email), using smart cards [9]. Another version of 2FA uses ambient noise to authenticate without users interference [10]. This eliminates the necessity of users waiting for SMS or email and entering the password, but since the surrounding noise can be mimicked by attackers users are advised to make unique sounds like coughing, singing/humming etc. As the authentication requires two keys , although passwords are leaked it is difficult for attacker to get the second key, hence users are safe even in case of password leaks. But because of the inconvenience process of waiting for the one-time password or confirmation code to arrive on SMS or email and ambient noise can be mimicked or the attacker can be present in the same location as the user, this method is not commonly used. Banking application is found to use 2FA despite having such user inconvenience as security is of primary importance compared to the ease of log-in.

III. **Server-side Measures**: By deploying techniques that detect password crack-

ing, optimize storage by manipulating hash of the password that decreases the success rate of inversion attack, using customized hardware to store and retrieve passwords, organizations take all measures possible to avoid password leaks.

- Various algorithms and techniques are developed to detect password leak / password cracking [11] [12]. These are often complex and costly to deploy and maintain. We have lately observed that despite having such methods in place attackers are successful in breaching the system .
- Hardware scrambling is another way of avoiding password leaks [13]. Trusted hardware such as Scramble S-Crib [14] is used to scramble passwords before they are stored in hash on a server. The hardware holds the encryption keys that scramble the password and one needs this hardware to do any password attack. Hence even in case of password leaks attackers cannot crack the passwords as they are scrambled.

The above mentioned methods do not guarantee the safety of an account in case of password leak , as strong passwords only require extra time or better mangling rules to crack they do not prevent attackers form accessing user accounts. OAuth can be vulnerable to an App Impersonation attack also this method cannot mitigate the adverse effects if the passwords are leaked from the service providers database. In reference to the above given example, OAuth would not be effective if the passwords are leaked from Twitter's database. SAuth and 2FA are effective to deal with password leaks situations but they add an additional step in the authentication process, which

is grievous to the users. S-Crib are costly to deploy and compromising the trusted hardware could allow attachers to access password database without even getting noticed, it does not provide any preventive measure after the passwords leak. There are no user side precautionary methods that reduce the harmful effects of password leaks, a simple method to achieve this is by changing a user's password soon after a password leak. Even better would be to change frequently users' passwords, so that users are protected even when a password leak occurs that they are unaware of.

Many websites and organizations (such as the OKEY password system) 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 becomes a problem for users who have many accounts with unique passwords to remember. Remembering many different passwords, changing them periodically, and keeping track of them is infeasible for the average user [15]. As a workaround for this problem, users use similar or the same passwords for multiple accounts [4]. Alternatively, they may change their passwords by only a character or two. Both of these options leave users vulnerable to a password breach, because compromising one account's password compromises all their accounts, as well as revealing patterns in their passwords that make them easier to predict in the future.

1.1 Outline of Contributions

We propose a solution in the form of an algorithm that automates password resets by leveraging a password manager to store completely random passwords, which

are secure with Shannon entropy value of 7-bits per character. We implement the proposed algorithm in a Firefox add-on, FluidPassword and demonstrate it to be working successfully on 63% of 41 websites which are considered from the top 100 websites [23]. We also suggest that this percentage can also be increased solving few difficult problem as mentioned in chapter 5 section 5.2

1.2 Outline of the Thesis

The rest of the document is organized as follows. In chapter 2 we discuss the background on passwords. In chapter 3 we discuss the motivation behind our work. In chapter 4 we detail the algorithm design and implementation. In chapter 5, we detail the results. Chapter 6 about the usability and security aspect of the implementation. In chapter 8 we discuss future work. Chapter7 discusses related work, and chapter9 gives the conclusion

Chapter 2

Background of Passwords

Password-based authentication is one of the most popular forms of authentication. A few services encrypt passwords for storage, which involves a strong, unique key for encryption that in most of the cases is stored in the same place as the passwords. The alternative is to store passwords in a cryptographically-secure hash function, which is a one-way function with no collisions that an efficient adversary can find. Moreover, the recommended practice is for passwords to be hashed with a salt, which is a prefix or postfix added to the password before it is hashed to prevent duplicate passwords from being stored as the same hash. In this case, the hashed output is stored along with its salt. The addition of salt is necessary to defeat attackers using rainbow tables, which are dictionaries of words and their hashes that allow attackers to significantly speed up their guessing ability.

Chapter 3

Motivation

Breaches that leak millions of passwords, e.g. those of LinkedIn [16] [17] [18], eHarmony, Facebook, Yahoo [19] [20], Amazon, Walmart [21], and iCloud [22] are becoming a daily news event. User accounts are compromised and users' personal information is at risk. Many of the leaks are not even confirmed until the attackers publish the passwords on-line. Organizations spend huge amount of money to prevent such attacks from being successful, but the availability of powerful password cracking platforms and the increase in complex and sophisticated exploitation software allow attackers to break into systems.

Many methods are being proposed to protect users from the effects of password leaks, but most of these require implementation on the organization's side. Moreover, despite the efforts of organizations, individual users have to be cautious too, by creating strong passwords and resetting their passwords often. We propose a simple and powerful method to protect users from the adverse effects of password leaks. The idea is to change users' passwords frequently, irrespective of the occurrence of password leaks, since those can only be identified after they have occurred.

Since users have multiple accounts, changing passwords on a frequent basis would be a tedious task. Although websites give strong password suggestions, studies have shown that most users do not follow them because they are hard to remember [4]. People also use similar, sometimes the same, passwords for multiple accounts to remember them. When they reset a password, they only make minor changes to their existing password. Thus, over a period we can observe a pattern in the passwords generated [15]. This makes it easy for an attacker to guess a user's password when they have enough information about their old passwords. We propose automating the process of a password reset without any user involvement. By automating the process and leveraging the functionality of password managers to save and retrieve passwords, users would remain secure without needing to remember their passwords, or remembering to change their passwords.

Chapter 4

Design & Implementation

4.1 General Overview

The application code flow is illustrated in figure 4.1. The steps are as follows:

I. Identify a login page

II. Capture the login credentials

III. After a successful login, if password reset URL is available select the URL and move to step 5. if the reset URL is not available search the current page for any possible URL′s to the password reset page.

IV. Select the URL with the highest priority. Priority values are assigned to the URL's depending upon the probability that the URL will lead to a password reset page. For example a hyper link which says "password change" would have high probability, than a hyper link which says "help" or "about us" and hence the algorithm selects the first URL.

V. Push the current page 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.

(b) If the new page is a login page, enter the captured credentials and return to step 3.

(c) If the new page is none of the above, search the current page for possible URL′s, returning to step 4.

VI. Close the extra tabs opened by the program.

4.2 Critical Steps

I. **Identifying the Login Page**

- Login pages can be identified by searching for a form with a pattern of text field, password field and a submit button.

II. **Capture Login credentials**

- Login pages can be identified by searching for a form (login form) with a pattern of a text field, password field and a submit button.

III. **Identifying Successful Login**

- Most of the websites usually redirect to the Login page if we fail to login; therefore a successful login can be identified by checking for the login form after entering the credentials.

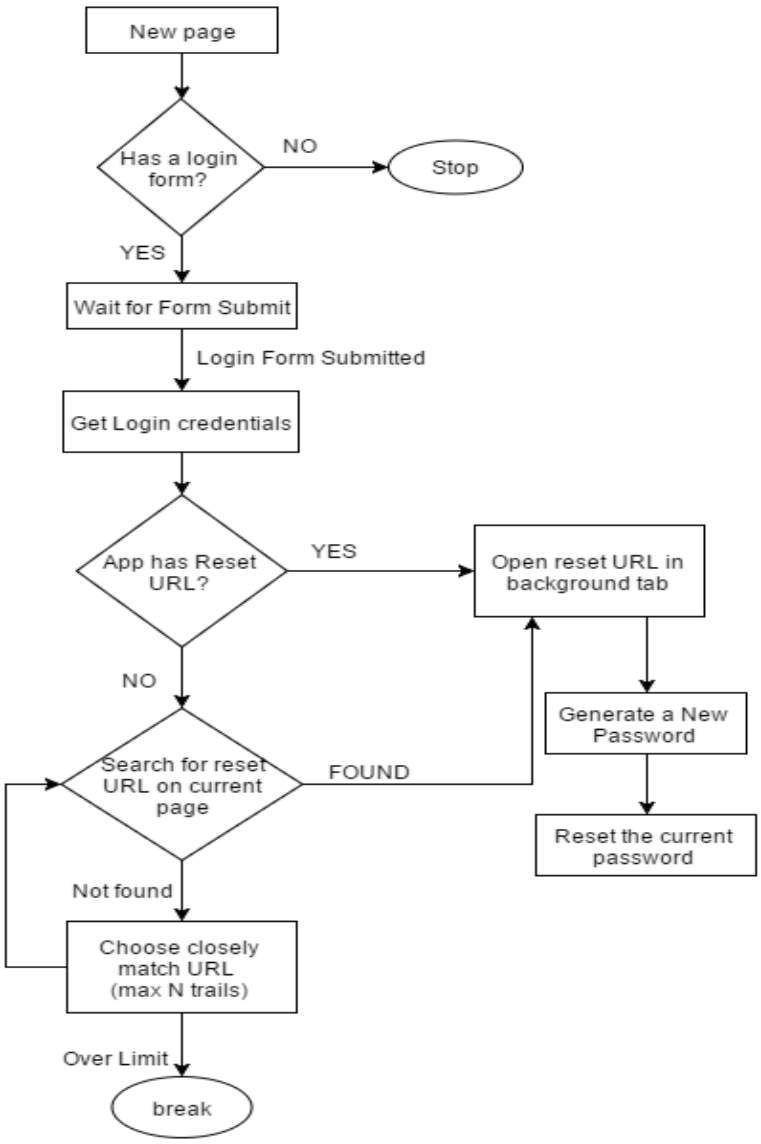


Figure 4.1: Algorithm - High Level

- We can also identify by searching for error text indicating login failure like “invalid password”,“incorrect username ” etc.

IV. **Selecting a URL**

- Each URL is assigned a priority level depending upon the probability that it would lead to the password reset page. For example a URL with “password change” pattern in it would have a high probability that it would lead to a password reset page, a URL with “setting” pattern would have less probability and URL with patterns such as “help” or “about us” would have no probability of leading to a password reset page. The current implementation has 10 priority levels as shown in Table 4.1. These patterns are selected based on observation of websites mentioned in Table 5.4.

  They are the most common links used to navigate for password reset. For example, on Facebook we navigate from *Settings* → *General* → *Edit Password*, on Amazon *Account* → *Change Account Settings* → *Edit Password* for Yahoo, *Account info* → *Account Security* → *Change Password*. Similarly we have other links such as *privacy*, *preference*, *my login* which could be used to navigate to change password page in other websites. Priority levels are assigned to these links depending upon the number of clicks required to reach to password reset page from that current page i.e.; lesser the number of clicks required to reach the password reset page, higher is the priority. As in the above examples for Facebook *Settings* is

Table 4.1: Priority and Patterns

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

assigned priority value 1, *Edit Password* is assigned a priority value of 8 (since it has "password" as substring), for Yahoo *Account info* is assigned a priority value of 3 and *Account Security* is assigned a priority value of 4 (since "security" is a substring). If there are more than one link present on current page which match the given list of patterns, we chose the one with highest priority. Therefore even though "settings" link is present on every page of Facebook, when we are on general settings page we choose "Edit Password" instead of "settings" since its priority is more.

- The URL with highest priority, available on the current page is selected. Once a URL is selected the current tab is pushed to 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 (pop from) and select for URL with next highest priority.

V. **Identifying a Password Reset Page**

- Password reset page is of various designs, but the most of them have a pattern of a username field (which is optional), three password fields and a submit button. We can identify a password reset page by searching for a form with this design.

4.3 Implementation

Since web browsers are most common interfaces we use to access our on-line accounts, we choose to develop a Firefox Add-on as a part of implementation.

4.4 Experimental Setup

- **Hardware**: Intel(R) Core(TM) i7-4510U processor, 8GB RAM
- **Software**: 64-bit Windows 8.1 Operating System, FireFox 42.0, Mozilla Firefox Addon-SDK 1.17 and JQuery 1.11.3

Chapter 5

Evaluation

5.1 Classification

5.1.1 Websites

The Firefox Add-on developed was tested against the top 100 websites [23]. A detailed list all the websites are given in the tables below. Table 5.2 gives the list of all the Top 100 websites from Alexa. Table 5.3 lists all the website that were not considered along with the reasons. Table 5.4 gives the list of all the websites considered and if the add-on supports these websites or not.

Top 100 websites			
Google.com	Instagram.com	Google.es	Indeeed
Facebook.com	Msn.com	Googleadservices.com	Cnn.com
Youtube.com	Microsoft.com	Netflix.com	Amazon.in
Baidu.com	Aliexpress.com	Amazon.de	Go.com

Yahoo.com	Amazon.co.jp	Stackoverflow.com	Google.co.id
Amazon.com	Google.co.uk	360.cn	Xinhuanet.com
Wikipedia.org	Reddit.com	Craigslist.org	Blogger.com
Qq.com	Ask.com	Tianya.cn	Google.com.au
Twitter.com	Google.fr	Diply.com	nytimes
Google.co.in	Google.com.br	Ok.ru	Bbc.co.uk
Taobao.com	Tmall.com	Google.ca	People.com.cn
Live.com	Onclickads.net	Alibaba.com	Cntv.cn
Sina.com.cn	Pinterest.com	Google.com.mx	Pixnet.net
Linkedin.com	Wordpress.com	Pornhub.com	Gmw.cn
Yahoo.co.jp	Paypal.com	Google.com.hk	Ebay.de
Weibo.com	Mail.ru	Naver.com	Google.pl
Ebay.com	Tumblr.com	Amazon.co.uk	Googleusercontent.com
Google.co.jp	Imgur.com	Ups	Dailymotion.com
Yandex.ru	Sohu.com	Xhamster.com	Google.co.kr
Blogspot.com	Xvideos.com	Rakuten.co.jp	Wikia.com
Vk.com	Google.ru	flickr.com	Chinadaily.com.cn
Hao123.com	Imdb.com	Kat.cr	Dropbox.com
T.co	Apple.com	github.com	Livedoor.jp
Bing.com	Google.it	Soso.com	Ebay.co.uk
Google.de	Fc2.com	Nicovideo.jp	Dailymail.co.uk

Table 5.2: Alexa Top 100 Websites

Table 5.3 gives the reasons for the list of websites which are not considered. The add-on can work on websites which are in English, since the hyper link patterns are matched against text patterns in English and hence cannot support websites in foreign languages. For example, it supports Amazon.com but not Amazon.co.jp. The add-on does not support websites which ask for security questions for password reset, it is difficult to automate such actions. On ethical grounds torrent and adult website are also not considered. Websites with no logins (news websites) or websites which serve as landing pages to other sites are not supported since they do not have any login.

Websites NOT considered (count: 71)	
Reason	List of websites with rank
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), Googleadservices.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)

Table 5.3: List of websites not considered

Table 5.4 lists the websites which are considered and provides the limitation for the websites for which it does not work. The limitations because of which the add-on does not support few websites is it fails to access the "Change Password" button on Google website, does not recognize reset in buttons and fails for websites where it cannot update the text box value.

Websites considered (count: 29)		
Limitations(6)	Unable to access "Change Password" button	Google.com(1)
	Reset Link in Button	Tumblr.com(42), Dailymotion.com(93), Dropbox.com(97)
	textBox.value doesn't work	Wordpress.com(39), Netflix.com(52)
Working(23)		Facebook.com(2), Yahoo.com(5), Amazon.com(6), Wikipedia.org(7), Twitter.com(9), Linkedin.com(14), Ebay.com(17), Vk.com(21), Instagram.com(26), Reddit.com(32), Pinterest.com(38), Paypal.com(40), Imgur.com(43), Imbd.com(47), Stackoverflow.com(54), Craigslist.org(56), Ups.com(67), Flickr.com(70), Github.com(72), Indeed.com(75), Fc2.com(82), Nytimes.com(90), Wikia.com(95)

Table 5.4: List of websites considered

Table 5.5 gives the CPU usage in percentage and Memory usage in MB during

the process of password reset, with and without the add-on.

	With Add-on		Without Add-on		Overhead	
Websites	CPU%	Memory(MB)	CPU%	Memory(MB)	CPU%	Memory(MB)
Facebook	11.11	117.95	10.37	108.56	0.74	9.9
Yahoo	10.72	156.3	10.19	140.32	0.53	15.98
Amazon	12.56	197.76	11.74	160.52	0.82	37.11
Wikipedia	9.51	130.41	9.20	125.65	0.31	4.76
Twitter	10.92	162.87	10.81	143.51	0.11	19.36
LinkedIn	8.61	125.62	8.23	122.18	0.38	3.44
eBay	17.80	219.86	13.76	201.65	4.04	18.21
Vk.com	11.58	180.18	11.42	160.35	0.16	19.83
Instagram	15.64	156.3	13.19	140.32	2.45	15.98
Reddit	13.52	150.3	10.72	133.32	2.8	16.98
Pinterest	15.21	114.3	9.46	111.20	5.75	3.1
Paypal	11.43	235.27	10.54	200.32	0.89	34.95
Tumblr	12.61	152.40	12.03	141.03	0.58	11.37
Imgur	13.71	112.53	10.90	86.52	2.81	26.01
Fc2	10.63	113.51	10.09	110.3	0.54	3.21
Stackoverflow	13.92	125.52	11.57	98.51	2.35	27.01
Craigslist	9.34	120.47	8.68	115.8	0.66	4.67
Ups	14.61	130.12	13.18	119.37	1.43	10.75
Flickr	10.51	162.3	9.43	153.4	1.08	8.9

Github	12.86	184.81	10.96	176.31	1.9	8.5
Indeed	11.89	96.53	10.34	92.04	1.55	4.49
Amazon.in	13.5	214.50	11.57	140.32	1.93	74.18
Nytimes	16.55	156.61	15.81	143.12	0.74	13.49
Wikia	12.75	96.3	11.63	91.32	1.12	4.98

Table 5.5: CPU & Memory Usage

The first two columns describe the CPU usage in percentage and Memory usage in MB by Firefox web browser during the process of password reset by the add-on for respective website. The third and fourth columns describe the CPU and memory usage by Firefox web browser while manually resetting the password for each web site. The last two columns give the over head of usage caused by the add-on. Although the values of add-on usage are slightly more for most of the cases, the difference negligible. The values clearly show that the load on CPU and Memory usage, using add-on is almost same as in the case of resetting password manually. The average overhead on CPU is 1.48% and the average memory overhead is about 16.54 MB. Also, the add-on makes a note of the password reset URL and hence can directly open the password reset page instead of going through the whole search process again. For example in the case of eBay, the algorithm searched about 10 pages before it lands on the password reset page. After successful completion of the process, the add-on makes a note of the password reset URL and hence when we run the same process again instead of searching through all the 10 pages it directly open the password

reset URL and, therefore, the load on CPU is much less when compared to manual password reset, since it is not possible manually to move to password reset page with one click on any website.

5.1.2 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.

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 password reset page is also simple with an optional username, three password fields and a submit button. Figure 5.1 and 5.2 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 5.3 shows a basic password reset page from Linkedin. The control flow in this website is straight forward right from identifying the login page to submitting the password reset form. For example: craigslist.com, facebook.com, reddit.com...etc, about 30.7% of websites supported (i.e; websites listed in table 5.4) fall under this category. The Initial implementation similar to the high-level design explained in figure 4.1 (without the stack implementation) successfully worked with these websites. We later had to modify the code for identifying they login page since

Figure 5.1: Login on landing page

few websites redirect or display the login page by clicking the "login" button.'

Websites with re-login

Websites that ask for a user to login again before they can change their password fall under this category. Websites like amazon, eBay etc. asks the user to enter their login credentials before they can move to the password reset page. About 19.3% of supported websites listed in table 5.4, fall under this category. To support these websites and additional check for login page was added. Once a selected URL was opened in new tab we check for a login page. If TRUE, login form was submitted using the captured credentials and after successful login it would redirect to password reset page.

Complex Websites

Priorities are assigned to URL's based on the probability that the URL would lead to password reset page and, therefore, it's not always certain that we would find the password reset page. The basic algorithm design would stop when the web page was not a password reset page, and the search process would not find any new links to move forward. Specifically in case of e-commerce sites such as amazon and eBay, as the web pages have a large number of hyper links. About 50% of supported websites

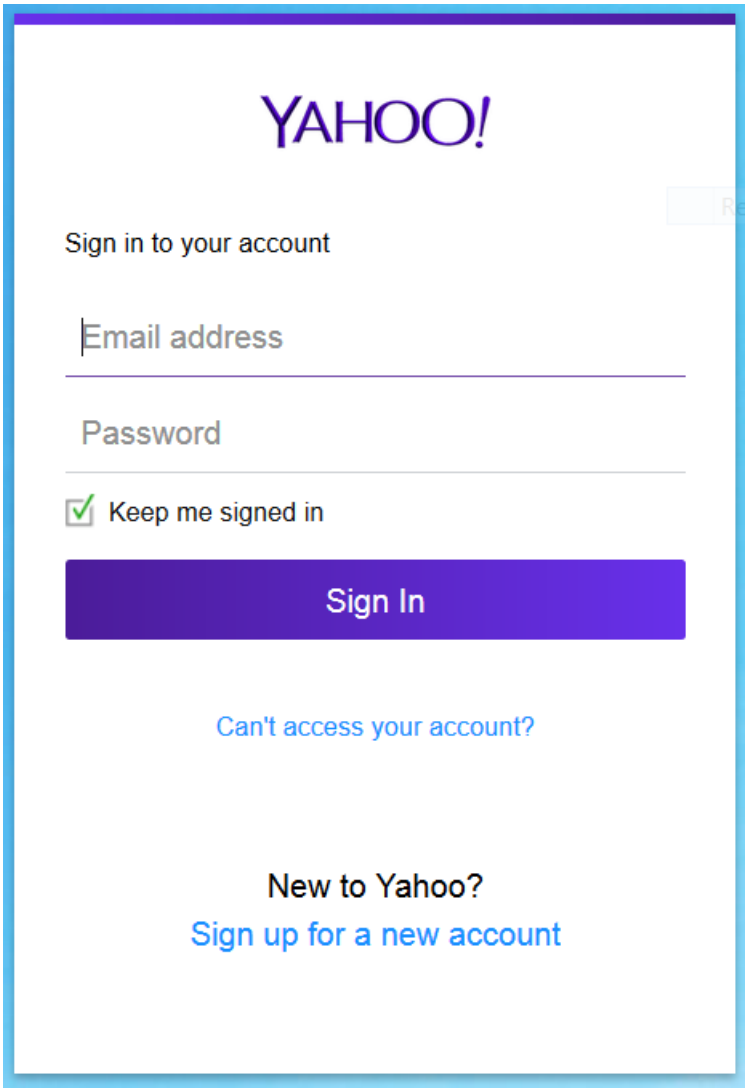


Figure 5.2: Simple Login page

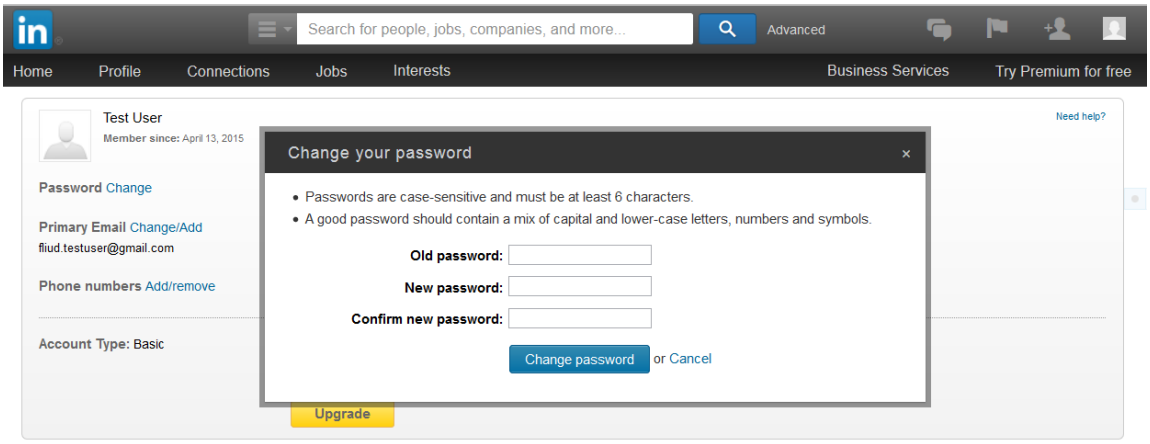


Figure 5.3: Simple Password reset page

(i.e; websites listed in table 5.4) fall under this category. Therefore to handle such scenarios a depth-first approach was implemented to move back to the previous tab and select the URL with next priority, in case the highest priority URL does not lead to any other link. Theoretically this design would exhaustively search all URL's available on the web pages until it finds a password reset page. However, the implementation places a threshold on the number of pages that can be opened, which is 20 i.e. if the code cannot find the password reset page in 20 searches, it terminates. Figure 5.5 gives the depth first design flow explained above.

5.2 Limitations

The current implementation does not support any Google website (Google.co.in, Youtube.com, Google.co.uk, Blogspot.com, Google.ca, Googleadservices.com, Blogger.com, Google.com.au, Googleusercontent.com), the only hurdle here is not being

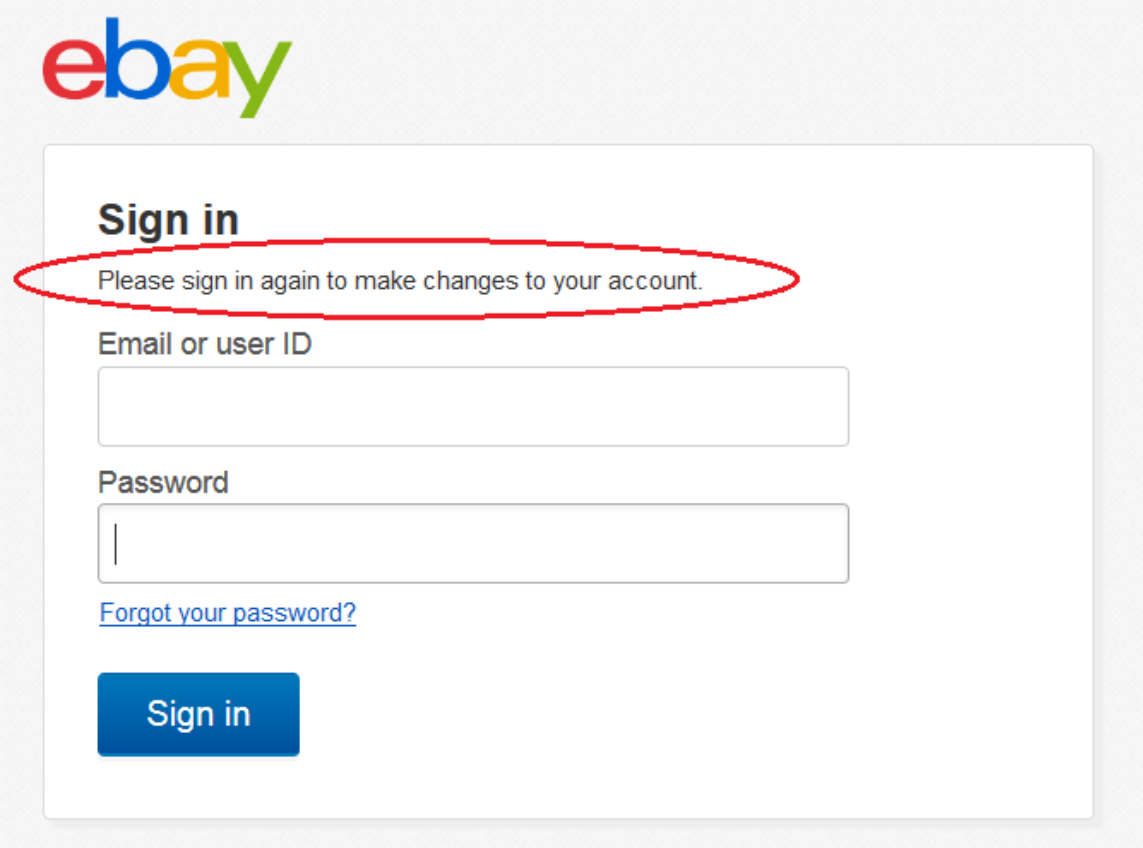
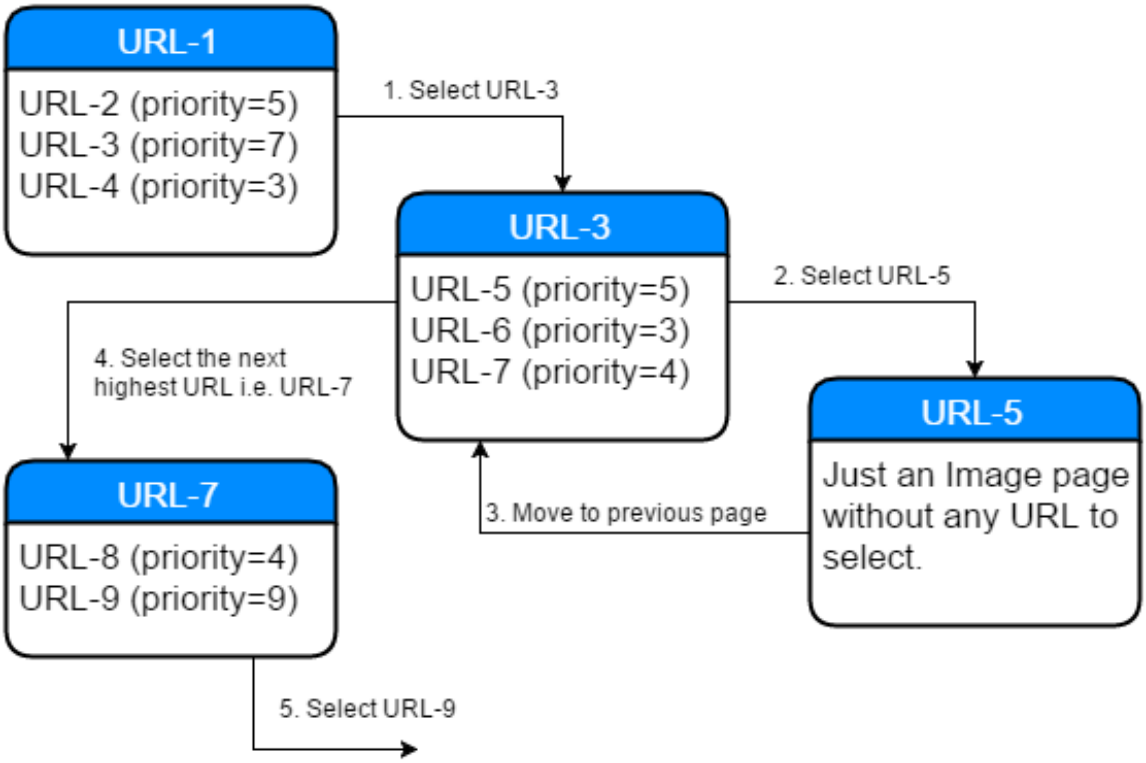


Figure 5.4: eBay Re-Login page



(1) Selecting URL-3 (highest priority level) and opening in new tab. (2) A New page with URL-3 has 3 URL's with URL-5 at the highest priority level. Therefore open URL-5 in a new tab. (3) Since the new tab has no new URL, we move back to the previous page. (4) Select the next highest priority URL (URL-7) and open in new tab. (5) Select the highest priority URL, and the process continues.

Figure 5.5: Depth-First design

able to access the "Change Password" button to submit the password reset form. This is difficult because this page does not have any forms rather everything are embedded in nested div's (including the button), by carefully scanning through div's on the page and identifying the innermost div which forms the button and clicking it could be a solution. We can clearly see that solving this problem would increase the supported websites from 81% to about 87%[1]. Similarly allowing the add-on to check for reset links in buttons instead of just text hyper links, would allow it to support more websites (Dropbox.com, Dailymotion.com, Tumblr.com) and increase the percentage to 92%. This can be achieved by scanning all the input elements on page with type = "button" and elements with tag <button>... </button>. The hindrance with wordpress and netflix is that we cannot simulate typing, a simple assigning of value to the text box (using textBox.value) does not work in this case. A solution to this could help add-on support more websites.

The add-on does not support websites that send a confirmation code or reset link to user's email/phone or ask for security questions that the user have selected during account creation. For example Bing, Msn, Alibaba, Aliexpress, etc. Many of the banking websites ask for Captcha as a necessary field for in login, add-on does not support such websites. As these actions require user intermission, it is hard to automate, since retrieving these details without users knowledge is a challenging task.

[1]Total no. of websites supported 23 + 3 (2 duplicates of amazon and 1 of ebay) = 26 out of considered, 32 (29+3) i.e; 81%. If we include Google websites, we increase the number to 35 out of 41, i.e; about 85%

Chapter 6

Discussion

6.1 Usability

The add-on is very simple to use just plug it into your firefox web browser just like any other add-on. One of the most important steps in setting up the add-on is to store, in your password manager, the login credentials for a site you want to use the add-on with. Since the add-on leverages the functionality of the Firefox password manager, it works only on websites that have user credentials stored. For example, if user A stores Facebook credentials in the password manager, but not his Linkedin credentials, then the add-on would only work with his or her Facebook account, not Linkedin. The add-on works similarly in the case of multiple users. For example, suppose we have two users, A and B, where A stores his or her Facebook password in the password manager and B doesn't. Then the add-on would change A's password every time he or she logs into Facebook, but would never change B's password.

Each time the add-on resets a user's password, it updates the password manager with the most current password. Therefore, users do not have to remember their

passwords. If required, they can retrieve their current password from the password manager. In case of any errors or exceptions the algorithm stops execution and does not reset the password.

However, the add-on does add some delay to the login process. This is because the add-on's listeners take sometime to attach to the web page elements, and they must wait until the web page loads all its content. Table 6.6 shows the increased duration of login process[1] for eBay.com with the add-on as opposed to without add-on. The sign-in process to eBay usually takes about 5 to 8 seconds (an average of 6 seconds), where users have to navigate to login page and click login button (we do not consider filling the login credentials as the password manager would auto-fill the fields). This same process would take about 12 to 15 seconds when using add-on since it takes about 2 to 4 seconds for the listeners to attach after the web page is loaded. This is because in case website designs similar to ebay, pinterest etc; where we have to click on "sign-in" or "login" to move to login page, the listeners are attached twice, first to the "sign-in" or "login" buttons or hyper links and next to the submit button on login page which require 2 to 4 second each, therefore a total of 4 to 8 seconds is added, which increases the duration of login process to about 15 seconds. In case of website like Facebook, listeners are attached only once and hence the delay is less (just 2-4 seconds). In other words its safe to wait for 12 to 15 seconds before we hit "Login" button for the add-on to work successfully every time. When the add-on's listeners fail to attach, or the user submits the form before

[1]Time from entering the website URL in browser to entring the login credentials and clicking "Login".

Table 6.6: Average delay is sign-in when using add-on

Website	Without Add-on	With Add-on
eBay.com	6 sec (avg)	15 sec (avg)

Table 6.7: Average duration to open and close 10 tabs

Add-on (on eBay)
13 sec (avg)

the listeners are attached, the add-on does not continue executing and fails to reset the user's password. In case of any error or exception the browser would not crash, it will work just as if the add-on was not installed. In Table 6.7 we see that it takes about 7 to 9 second to open ten random tabs and close them one by one, and it takes about 12 to 15 seconds for the add-on to scan through web pages until it finds the password reset page with eBay[2].

Once the add-on has successfully reset the password, it stores the URL of the password reset page in a separate file named "purls.txt", this is a file where the website host link and the password reset link are stored in the form of a key value pair. After that, whenever it encounters the same website, it uses the link in "purls.txt" to go directly to password reset page. This results in a password reset even faster than if a user resets their password manually.

[2]We chose eBay because it takes the highest number of web pages for add-on to scan until it reaches password-reset page, which is 10.

6.2 Security

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

The add-on uses Stanford Javascript Crypto Library (sjcl.js) as a secure pseudo-random number generator to generate secure, random 12-character 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.

Since passwords are chosen probabilistically, we can define 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 6.1 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.

$$
\begin{aligned}
H(X) &= -\sum_{i} \mathrm{P}(x_i)\log_b \mathrm{P}(x_i) \\
&= -\sum_{i=1,\ldots,94} \frac{1}{94}\log_2\left(\frac{1}{94}\right) \\
&= -\frac{94}{94}\log_2\left(\frac{1}{94}\right) \\
&= \log_2(94) \\
&= 6.55458885168...
\end{aligned}
\tag{6.1}
$$

RFC 4086 recommends 56 bits for higher-than-moderate-security passwords in 2015[3] [24]. 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.

[3]They recommend 49 bits, then later recommend adding 2/3 of a bit per year due to Moore's law, giving us 56 in 2015.

[4]75 bits (for a password in 1995) $+20 * 2/3$

Chapter 7

Related Work

Password survey: Several surveys conducted on variety of user domains such as professional users [25], academic students [4] [26], common public [27] to assess their attitude and awareness towards password security and related threats. An in-depth study of users approach towards password composition, reuse and management practices reveal that the knowledge of password security and effects of weak passwords play a crucial role [26], and common public generally tend to use less complex as little characters as possible [28] or use related/similar pattern passwords [25]. While few others employ other techniques such as maintaining diaries, protected files to keep track of the moderately complex passwords across different websites, which could be devastatingly harmful if the protected files are compromised.

Although stringent password generation rules [25] [29], pre-generated/passphrases suggestions [30] are employed to make sure strong passwords are used, studies show that majority of the users do not adopt these practices [4] as they are difficult to memorize. Expiration of passwords after a certain period, instead of reducing predictability of passwords actually increases it when users keep creating similar pass-

words [4] [31]. Also, users tend to create similar passwords across websites which could result in a domino effect [32] and studies show that cross-site password guessing algorithms [31] are developed to take advantage of a known password from one website and easily guess the user's passwords on other websites.

Password Reuse: Increase in the e-commerce application has led to the increase in the number of accounts an individual user holds, implying an increase in the number of passwords to maintain in which case users tend to use same or similar passwords on all websites. In such as situation the password is no stronger than the weak system using that password. Since if that system is hacked, hackers can potentially infiltrate other websites [32]. Single sing-on and OPass [5] are few alternatives that encourage password reuse. Single sing-on uses a trusted web service for authentication on all applications and OPass adds an additional one time key, sent the user's cell phone along with the password.

Other Alternatives: Various other authentication protocols have been suggested such as S-Auth [7], O-Auth [6], and 2FA (2 Factor Authentication) [9] [8]. S-Auth employs authentication synergy where it upon requesting access to a service it redirects to another trusted service and only when the user successfully authenticates on both services, access is provided. O-Auth allows a third party service to access the credential-restricted resource without the knowledge of the owner's credentials. In 2FA, users requesting for access are provided with a one-time password or a security pin number through SMS on their registered phone or email. This is used along with the actual password of the account in order to successfully authenticate. There have other improvements in 2FA to eliminate the user interference and allow

the users phone and laptop to identify each other and grant access [10]. Apart from the authentication protocols, hardware solutions [13] on the server side have also been suggested which require a trusted hardware to scramble the passwords before they are stored in database. This additional hardware is required every time we have to work with passwords i.e. every time a user tries to login.

In this paper, we propose a simple and easy solution of using Fluidpassword add-on in comparison with the above. It requires no change in the authentication process nor does it require any hardware implementation on the server side. Also, this is the only solution so far suggested which deals with password leaks from individual users side.

Chapter 8

Future Work

Websites are designed in various ways. Successful execution of the critical steps, as explained in section 4.2 are important for the algorithm to work. Capturing Login credentials is difficult for few websites where the web page have dynamic forms, forms nested in iframes, etc. (example: zillow.com, etsy.com, buzzfeed.com). Identifying a successful login becomes difficult when we have hidden or dummy login-forms (example: godaddy.com has a dummy login form on the user's profile page) or when it is difficult to identify a login page (example: aliexpress.com, alibaba.com, flipkart.com).

Traversing through URL's is the next crucial task to reach password reset page. Current implementation looks for text hyper links on the page to assign priority, and this method fails when the URL's are embedded as hyperlinks in images or if the links are setup in a button and it needs to be clicked. In addition to this identifying a password reset page is also important, it is tough when the page is not designed in a regular pattern i.e. Optional username, three password fields and a submit button (example: wordpress.com, themeforest.net)

Improvements can be done to the scripts, written for above mentioned tasks such that the algorithm would be able to handle different web designs and also decrease the initial login delay. Websites that ask for security questions when requested for password reset can also be handled if we can setup the add-on to store a set of questions with answers and modify the code to check for the questions and them answer accordingly from the stored information. In addition, a mobile version of this add-on can be developed, and FireFox Sync can be utilized to synchronize the Firefox browser on the personal computer and mobile device.

Chapter 9

Conclusion

We have developed *Fluidpassword* Firefox add-on as an implementation of our algorithm which automates password reset. Users only need to install this add-on on their Firefox browser and the password would be reset[1] each time they successfully login into their accounts. Additionally passwords are reset much faster from the second attempt when compared to resetting them manually, as the add-on keeps a note of the reset URL when it successfully resets password for the first time and hence move the password reset page directly. Since users don't need to remember the passwords, the algorithm generates strong random passwords, whose entropy is currently (84bits) is more than the suggested (56bits) according to RFC 4086. This also mitigates password reuse problems since there is no user involvement. Users can be safe from the adverse effects of password leaks as the passwords change frequently.

While there are many more improvements that can be done as mentioned in Chapter 8, this is by far the only precautionary solution suggested from individual's side to deal with password leaks.

[1]Only for the websites whose credentials are stored on password manager

Bibliography

[1] E. Grosse, "Gmail account security in Iran," 2011. [Online; accessed 8-September-2011].

[2] M. Honan, "How Apple and Amazon securiy flawss led to my epic hacking." `http://www.wired.com/2012/08/apple-amazon-mat-honan-hacking/`, 2012. [Online; accessed 6-August-2012].

[3] V. J. Yodaiken, "Systems and methods for detecting a security breach in a computer system," Sept. 8 2009. US Patent 7,587,763.

[4] M. Farcasin 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.

[5] 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*, pp. 271–276, IEEE, 2013.

[6] 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*, pp. 271–278, ACM, 2014.

[7] 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*, pp. 187–198, ACM, 2013.

[8] B. Selvarajan, "Simple two-factor authentication," May 10 2007. US Patent App. 11/267,148.

[9] 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*, vol. 74, no. 7, pp. 1160–1172, 2008.

[10] N. Karapanos, C. Marforio, C. Soriente, and S. Capkun, "Sound-proof: Usable two-factor authentication based on ambient sound," *arXiv preprint arXiv:1503.03790*, 2015.

[11] N. Chakraborty and S. Mondal, "A new storage optimized honeyword generation approach for enhancing security and usability," *CoRR*, vol. abs/1509.06094, 2015.

[12] H. Mohammed, C. N. Gutierrez, M. J. Atallah, and E. H. Spafford, "Ersatz-passwords: Ending password cracking and detecting password leakage,"

[13] S. crib, "Scramble SCrib - No More Password Leaks." `http://www.newswire.com/scramble-scrib-no-more-password/251627`, 2013. [Online; accessed 28-Nov-2013].

[14] 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].

[15] 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.

[16] Wikipedia, "2012 linkedin hack — wikipedia, the free encyclopedia," 2015. [Online; accessed 14-November-2015].

[17] 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].

[18] S. M. Kelly, "LinkedIn Confirms, Apologizes for Stolen Password Breach." http://mashable.com/2012/06/06/linkedin-passwords-hacked-confirmation/#DnD4MWqOTuq3, 2012. [Online; accessed 06-June-2012].

[19] H. 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-2d1169163 2013. [Online; accessed 04-December-2013].

[20] C. Hubbell, "Huge Yahoo Password Leak 453,441 Passwords Exposed." https://geeks.online/

huge-yahoo-password-leak-453441-passwords-exposed/, 2012. [Online; accessed 12-June-2012].

[21] 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].

[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-isnt-only-problem-apples-backup-service, 2014. [Online; accessed 02-September-2014].

[23] "Top 100 websites." http://www.alexa.com/topsites, 2015. [Online; accessed Feb, 2015].

[24] D. E. 3rd, J. Schiller, and S. Crocker, "Randomness Requirements for Security." RFC 4086 (Best Current Practice), June 2005.

[25] A. Adams and M. A. Sasse, "Users are not the enemy," *Communications of the ACM*, vol. 42, no. 12, pp. 40–46, 1999.

[26] G. B. Duggan, H. Johnson, and B. Grawemeyer, "Rational security: Modelling everyday password use," *International journal of human-computer studies*, vol. 70, no. 6, pp. 415–431, 2012.

[27] K. Bryant and J. Campbell, "User behaviours associated with password security and management," *Australasian Journal of Information Systems*, vol. 14, no. 1, 2006.

[28] A. M. De Alvaré, "How crackers crack passwords or what passwords to avoid," tech. rep., Lawrence Livermore National Lab., CA (USA), 1988.

[29] P. G. Inglesant and M. A. Sasse, "The true cost of unusable password policies: password use in the wild," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 383–392, ACM, 2010.

[30] R. Shay, P. G. Kelley, S. Komanduri, M. L. Mazurek, B. Ur, T. Vidas, L. Bauer, N. Christin, and L. F. Cranor, "Correct horse battery staple: Exploring the usability of system-assigned passphrases," in *Proceedings of the eighth symposium on usable privacy and security*, p. 7, ACM, 2012.

[31] A. Das, J. Bonneau, M. Caesar, N. Borisov, and X. Wang, "The tangled web of password reuse," 2014.

[32] B. Ives, K. R. Walsh, and H. Schneider, "The domino effect of password reuse," *Communications of the ACM*, vol. 47, no. 4, pp. 75–78, 2004.

VITA

Akhileshwar Guli

Candidate for the Degree of

MASTER OF SCIENCE

Thesis: FLUIDPASSWORDS - MITIGATING THE EFFECTS OF PASSWORD LEAK AT USER LEVEL

Major Field: Computer Science

Biographical:

Education:

Completed the requirements for the Master's of Science degree with a major in Computer Science at Oklahoma State University in December, 2015. He holds a Bachelor of Science in Information Technology from Jawaharlal Nehru Technological University, Hyderabad from 2011.

Experience:

Software Engineer Worked with Birlasoft INDIA Ptv. Ltd for 2 years from 2011 to 2013 as Oracle EBS Developer.

Graduate Assistant Computer Science Department, Oklahoma State University, Stillwater, OK: iMac Lab, Spring 2014.

Teacher's Assistant Computer Science Department, Oklahoma State University, Stillwater, OK: Discrete Math and Computer Systems-I, Fall 2015.

Research Assistant Networks Lab, Oklahoma State University, Stillwater, OK: Spring 2015.