SIJITCS Vol.: 03 II Issue I II Pages 01-06 II Jan 2019

Proposal of a Security Solution for Preventing Password Autofill in Specific Phishing Web Pages

Somchai Chatvichienchai
Department of Information Security, Faculty of Information System, University of Nagasaki
somchaic@sun.ac.jp

Abstract— Passwords are the primary authentication method of the web services. Password managers, which are provided by web browser vendors and third party software vendors, relief burden of remembering distinguished password of each web service. However, password managers contain so much confidential information that is valuable to cyber criminals, password manager is often the target of hackers. Much effort has been done for defending confidential information stealing problems such as the attack of the tracking script, etc. Recently, new phishing pages that deceive password managers to autofill confidential information into the fields that are invisible to the naked eyes. The experiment which is done by this work shows that the above phishing pages can be used to steal stored passwords from browsers' password managers. This problem has not yet been solved by web browser developers. The objective of this article is to propose a security solution for preventing password autofill in these phishing pages. The proposed solution turns off autofill function of the web browser that the user is using. Based on analysis of HTML source of login page, the proposed solution informs the existence of hidden fields and has the user confirm whether to permit the web browser to perform auto-filling on the login page.

Index Terms— autofill, password, solution, web.

I. Introduction:

Passwords are the primary authentication method of the web services. The previous works [1], [2] indicate that a user tends to choose bad passwords and/or reuse passwords over multiple sites. This behavior increases the potential damage if a password is stolen, cracked, or if a service that has access to it is compromised, since the attacker will be able to reuse it on all online services that share the same password. In order to relief burden of remembering a password of each web service, browser vendors as well as third party software vendors have developed password managers capable of storing these secret credentials for the users. When the users visit a webpage and fill out a login form, browser's password manager asks the users if they want to save the login details. Since browser's password manager automatically fills in login information based on the current domain of the web page, it provides some protection against typo-squatting and phishing attacks [3], [4].

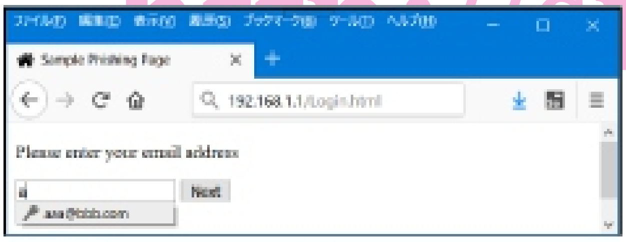
Since browser's password manager contains so much confidential information (such as email address and passwords, etc.) that is valuable to cyber criminals, it is often the target of hackers. Some existing works [5], [6] have demonstrated how an attacker injects a tracking script written by JavaScript into web pages that are vulnerable to XSS (cross site scripting) attacks. When the user browses that web page, the tracking script inserts a malicious login form that is invisible to the naked eye onto the webpage, and password manager automatically fills in the user's login information of the website that the user is visiting. By this way, the tracking script snaffles up user's login information from the invisible form's field and sends the login information in form of a hash value to the attacker's server. The most convenient feature of password manager of web browser is also one of the weakest links in its security.

Some existing works focus on confidential information leak due to autofill of browsers. However, there exists no work providing a perfect prevention against the attack of the tracking script. Silver et at. [5] proposed a defense method that always requires some user interaction before auto-filling a form. Their proposed method will prevent sweep attacks where multiple passwords are extracted without any user interaction. User interaction can come in the form of a keyboard shortcut, clicking a button, selecting an entry from a menu, or typing into the user name field. However, the defense method can't prevent browsers from auto-filling password into the hidden field of the new phishing page. After the user clicks the submit button, the secretly auto-filled password will be sent to the third party. The scenario of the password attack will be presented in the next section.

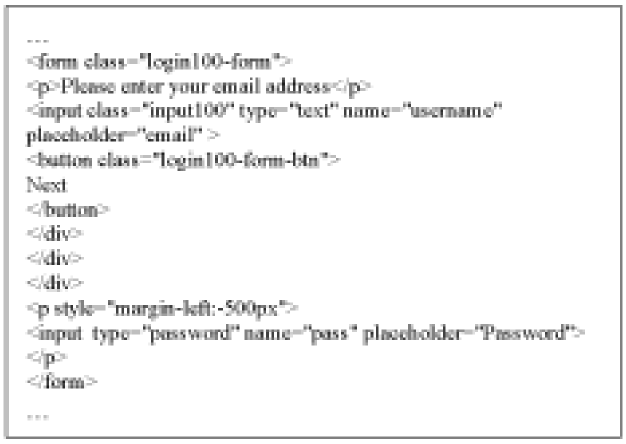
The objective of this article is to propose a Browser Autofill Control System to prevent undesired autofill of passwords in these phishing pages. The proposed system disables the autofill feature of the web browser that the user is using. Given a URL address of the login web page, the proposed system analysts HTML data of the login page and display the names or IDs of fields that are the target of autofill. By this way, the user can recognize dangerous hidden fields and can decide whether or not to permit the browser to perform autofill on the login page.

II. Scenario of Password Stealing:

In 2017, Gibbs has posted an article [7] describing a new threat on password managers. The article claims that many web browsers, including Google's Chrome, Apple's Safari and Opera, as well as some plugins and utilities such as:



(a)



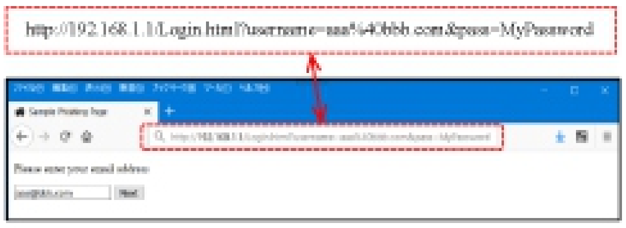
```
...
<form class="login100-form">
<p>Please enter your email address</p>
<input class="input100" type="text" name="username"
placeholder="email" >
<button class="login100-form-btn">
Next
</button>
</div>
</div>
</div>
<p style="margin-left:-500px">
<input  type="password" name="pass" placeholder="Password">
</p>
</form>
...
```

(b)

Fig 1.(a) A sample screenshot of a phishing webpage before a user inputs email address, and
(b) a part of HTML source of the phishing webpage.

LastPass, can be tricked into giving away a user's personal information through their profile-based autofill systems. He has revealed how such attacks work on a demonstration phishing page [8]. The phishing page has a form in which only the fields "name" and "email" can be seen, along with a "send" button. However, the source code of the web page harbors some hidden confidential information from the user: there are six other fields (phone, organization, address, postal code, city and country), which the browser also automatically populates if the user has activated the autofill function. These six fields cane not be noticed by the user since these fields are defined as static inputted textboxes whose positions are set by negative margins.

In this work, an experiment using the above technique to steal passwords from password managers of browsers has been done. The result of experiment shows the successfulness of password stealing. Figure 1 (a) depicts a screenshot of a sample phishing webpage which displays only the email textbox and a button for next operation. This phishing page can be implemented into the website where users have already saved their passwords by password managers. The implementation can be done internal offenders of the website or by hackers who can penetrate security protection of the website. Figure 1 (b) shows a part of HTML source of the phishing webpage. Notice that password textbox is defined by the second input tag. However, the password's textbox doesn't appear in Fig.1 (a) since its position is outside screen area of web browser according to *<p style="margin-left:-500px">* tag. Note that *<p style="margin-top:-500px">* tag can also be used to make password's textbox disappear from screen area of web browser. This trick can also be done by using CSS (Cascading Style Sheets). As shown in Fig. 2, URL address bar outputted "*http://192.168.1.1/Login.html ?username=aaa%40bbb.com:*



http://192.168.1.1/Login.html?username=aaa%40bbb.com&pass=MyPassword
Sample Phishing Page
Please enter your email address
aaa@bbb.com
Next

Fig 2. A sample of the phishing webpage of Fig.1 (a) after the "Next" button was clicked.

&pass=MyPassword" after the "next" button is clicked.

It includes user's password that is automatically filled by password manager of web browser. Password managers autofill the password as far as this phishing page is implemented in the web server which is under the same domain name that is recorded by password managers.

What makes this scenario tricky is that internal offenders of the website or hackers add the third-party scripts to the web page, making it as a part of the website's own code. The web browsers' built-in protections, which isolate external third-party scripts from the site's code, don't work in that case. Therefore, web browsers have no mechanism to handle this problem. However, it's very surprising that this serious problem has not yet been fixed after Gibbs's article opened to public in 2017. At this point, there doesn't appear any solution to this problem other than turning off autofill function of user's browser. In the meantime, if the users decide to leave autofill turned on due to its general convenience factor, they'll need to be even more diligent about making sure they're only visiting known and trusted websites.

III. The Architecture of the Proposed System:

In order to prevent password managers from autofill of passwords on the phishing webpages described in the previous section, this article proposes *Browser Autofill Control System* (*BACS*, for short). The system is developed to be an interface between users and web browsers. The architecture of BACS is shown in Fig.3. A user will login web page thru BACS. After the system is started by users, it will disable autofill features of web browsers of the user's computer. The system has *Login Page Database* (*LPDB* for short), which stores data records each of which consists of a URL address and a hash value of HTML data of the login page which has been confirmed by the user. In case the login page is defined to work with a CSS file, a hash value of the CSS file will also be added into the record. The reason of storing these data in form of hash values is to decrease the:

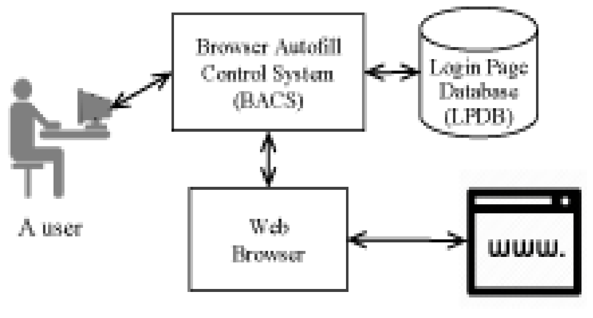

Browser Autofill Control System (BACS)
Login Page Database (LPDB)
A user
Web Browser
www.

Fig 3. The architecture of the proposed system.

burden of storing and comparing these data. The hash value of HTML data of LPDB record is used to compare with the hash value computed from HTML data of the login page accessed by the user in order to justify whether they are the same content. A hash value is computed by a hash algorithm [9]. The system employs a secure hash algorithm 256 (SHA256, for short) which is adopted as one of the most secure ways to protect digital information. SHA256 converts a data string of HTML data of a login page into an output 256-bit string. The output string is generally much smaller than the original data. SHA256 is designed to be collision-resistant [10], meaning that there is a very low probability that the same 256-bit string would be created by different data.

As shown in Fig. 3, BACS displays all input textboxes (including hidden textboxes) in the login page so that the user can decide whether to allow password manager of the browser to automatically fill the stored password and other values to these textboxes. If the user decides to allow autofill, the system will enable autofill of the browser and will restart the browser to read the login page.

IV. Methodology of Preventing Password Stealing:

Figure 4 illustrates the flowchart of justifying whether autofill of the web browser should be permitted. When BACS is installed, the user declares which browser will be controlled by the system. At diagram 1, the system disables autofill function of the browser. The next section will explain methods that disable autofill functions of some major browsers. Given a URL address of a login page that a user wants to read, the system (see diagram 2) instructs the browser to read the login page and associated CSS file (if any). As shown in diagram 3, the system checks the existence of input tags of textboxes in the web page. If there exist input tags of textboxes defined in the webpage, the system computes hash value of the HTML data (see diagram 4). In case the HTML data is associated with a CSS file, the system also computes hash value of the CSS file.

As shown in diagram 5, the system extracts the domain name from the URL address and uses it as a key to find the corresponding record from LPDB. Existence of a LPDB record whose hash values are the same as those of diagram 4 denotes that the user have already permitted auto-filling of the login page. Therefore, the system automatically enables autofill of the browser. In diagram 9, the system restarts the browser to make autofill feature of the browser to become effective. Thereafter the system has the browser opened the login page.

At diagram 6, if the system found that there exist no LPDB record having the specified domain name or there exist a record having the specified domain name but its hash values are not the same as those computed at diagram 5, it displays the names or IDs of input tags of textboxes of the login page. As shown in diagram 7, the user checks whether there exist hidden fields by comparing the fields that the system reported and those that appear in screen of the browser. If the user found that there does not exist hidden password field, she permits the autofill of the browser. In case there exist no record having the same domain name, the system adds a new LPDB record with the domain name and its corresponding hash values. Otherwise, the system updates the LPDB record with the new hash values computed at diagram 4 (see diagram 8). The system proceeds to the process of diagram 9.

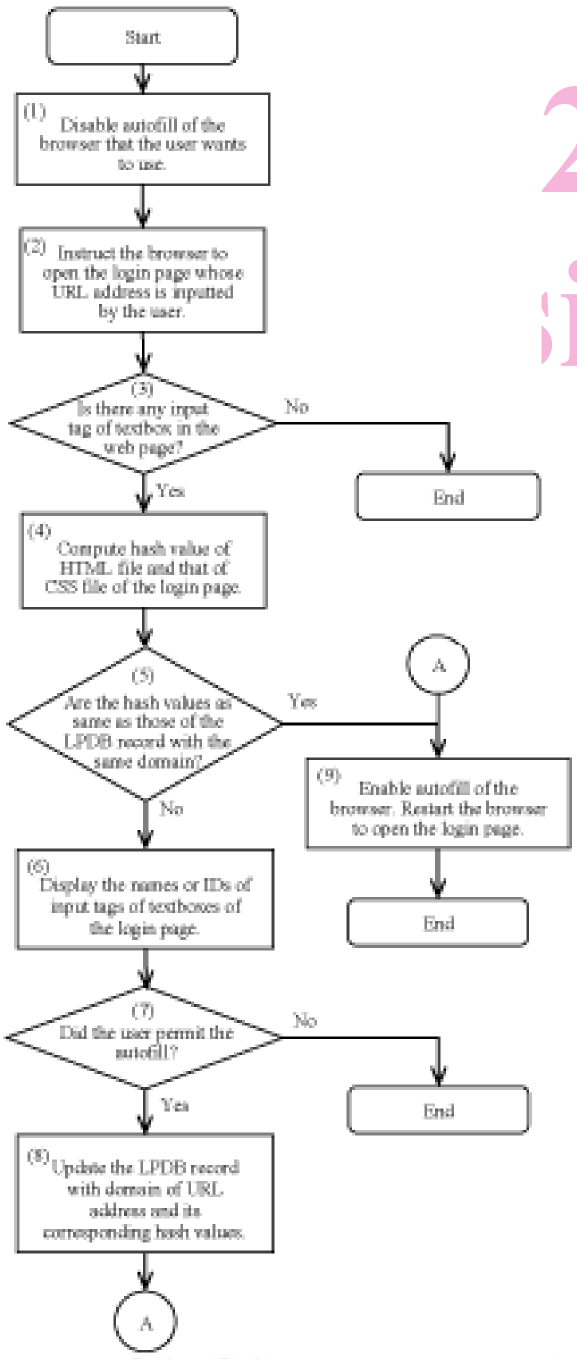
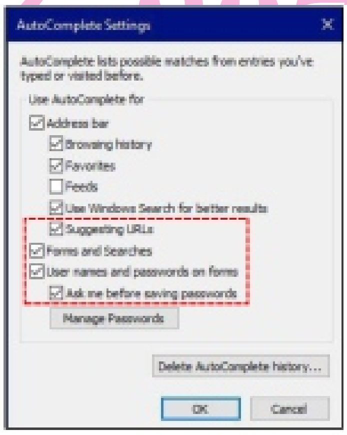

Start
(1) Disable autofill of the browser that the user wants to use.
(2) Instruct the browser to open the login page whose URL address is inputted by the user.
(3) Is there any input tag of textbox in the web page?
No
End
Yes
(4) Compute hash value of HTML file and that of CSS file of the login page.
(5) Are the hash values as same as those of the LPDB record with the same domain?
Yes
A
(9) Enable autofill of the browser. Restart the browser to open the login page.
End
No
(6) Display the names or IDs of input tags of textboxes of the login page.
(7) Did the user permit the autofill?
No
End
Yes
(8) Update the LPDB record with domain of URL address and its corresponding hash values.
A

Fig 4. The flowchart of justifying whether autofill on the login page should be permitted.

V. Methodology for Disabling and Enabling Autofill of Web Browsers:

This section introduces methods that disable and enable the autofill (in other word, autocomplete) feature of Internet Explorer (IE, for short) and Mozilla Firefox (Firefox, for short) without user intervention. Both web browsers use the option "autocomplete" to remember the typed text in a web form.

Figure 5 shows "AutoComplete Settings" of IE version 11 which autocomplete feature is enabled. In order to disable this


AutoComplete Settings

AutoComplete lists possible matches from entries you've typed or visited before.

Use AutoComplete for
- [✓] Address bar
  - [✓] Browsing history
  - [✓] Favorites
  - [ ] Feeds
  - [✓] Use Windows Search for better results
  - [✓] Suggesting URLs
- [✓] Forms and Searches
- [✓] User names and passwords on forms
  - [✓] Ask me before saving passwords

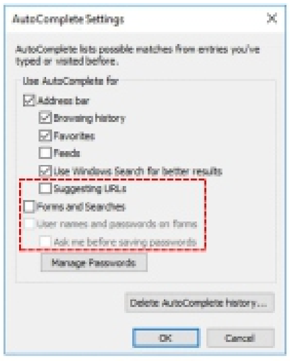
Fig 5. A sample screenshot of autocomplete settings of IE11 showing that autocomplete feature is enabled.

```
reg add "HKCU\Software\Microsoft\windows\CurrentVersion
\Explorer\AutoComplete" /v AutoSuggest /t REG_SZ /d No
reg add "HKCU\Software\Microsoft\Internet Explorer\Main" /v
"Use FormSuggest" /t REG_SZ /d No
reg add "HKCU\Software\Microsoft\Internet Explorer\Main" /v
"FormSuggest Passwords" /t REG_SZ /d No
reg add "HKCU\Software\Microsoft\Internet Explorer\Main" /v
"FormSuggest PW Ask" /t REG_SZ /d No
reg delete "HKCU\Software\Microsoft\Windows\CurrentVersion
\Internet Settings" /v DisablePasswordCaching
```

Fig 6. An example of a set of registry scripts for disabling autocomplete feature of Internet Explorer.

feature without human intervention, the system executes the set of registry scripts shown in Fig. 6. The system executes these scripts by using methods and properties of WScript object [11]. *CreateObject* method of WScript object is one of the most important functions in WSH script that can manipulate various components of the Windows environment.

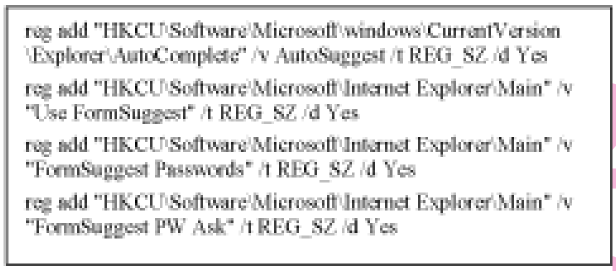
As shown in Fig. 6, the first script disallows IE to show a list of matching URLs in address bar, if the user has accessed the website earlier. This script is necessary to prevent IE from


AutoComplete Settings

AutoComplete lists possible matches from entries you've typed or visited before.

Use AutoComplete for
- [✓] Address bar
  - [✓] Browsing history
  - [✓] Favorites
  - [ ] Feeds
  - [✓] Use Windows Search for better results
  - [ ] Suggesting URLs
- [ ] Forms and Searches
- [ ] User names and passwords on forms
  - [ ] Ask me before saving passwords

Fig 7. A sample screenshot of autocomplete settings of IE11 showing that autocomplete feature is disabled.



```
reg add "HKCU\Software\Microsoft\windows\CurrentVersion
\Explorer\AutoComplete" /v AutoSuggest /t REG_SZ /d Yes
reg add "HKCU\Software\Microsoft\Internet Explorer\Main" /v
"Use FormSuggest" /t REG_SZ /d Yes
reg add "HKCU\Software\Microsoft\Internet Explorer\Main" /v
"FormSuggest Passwords" /t REG_SZ /d Yes
reg add "HKCU\Software\Microsoft\Internet Explorer\Main" /v
"FormSuggest PW Ask" /t REG_SZ /d Yes
```

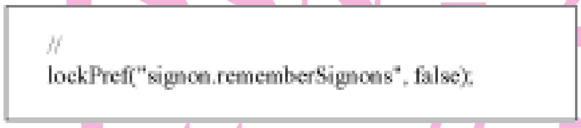
Fig 8. An example of a set of registry scripts for enabling autocomplete feature of Internet Explorer.

auto-filling the URL address that includes a user-id and a password. The second script disallows IE to call previously typed inputs of a field on the web forms. Therefore IE will

not suggest from the previous entries existing in the list, if any. The third script disallows IE to call the saved passwords on web forms. The fourth script disallows IE to prompt to save passwords on web forms. Therefore, the user can't click the saved password the next time the user visits the page. The last script deletes a windows registry key that disables password caching. Therefore, the user can have the browser saved the password that she has just inputted. In other word, the user can still change "user name and password on forms" and can prompt to save the password. Figure 7 shows AutoComplete Settings of IE version 11 after all the registry scripts of Fig. 6 have already been executed. Figure 8 shows a set of scripts that should be executed in order to enable Autocomplete feature of IE version 11.

Now let's consider methods of disabling and enabling autocomplete feature of Firefox. Firefox has the ability to customize installations using a configuration file called *mozilla.cfg*. This file should be created in the Firefox install directory, and each time Firefox is loaded it's checked for any custom configurations that have been added.

In order to disable autocomplete feature of Firefox, the system uses the mozilla.cfg file to lock *signon.rememberSignons* to false [12]. This is done by creating the mozilla.cfg file (if does not exist) and adding the following two lines into the file.



```
//
lockPref("signon.rememberSignons", false);
```

Figure 9 illustrates an example of scripts that add the above two lines into the mozilla.cfg file. To enable autocomplete feature of Firefox, the system deletes the above two lines from the mozilla.cfg file. Firefox should be restart in order to make the update effectively.

VI. Related Work:

Most research in the area of password managers focused mainly on three different aspects: generating pseudo-random and unique passwords for each single Web application based one some master secret [13], storing passwords in a secure manner [14] and protecting users from phishing attacks [15]. Using XSS attacks for stealing autofilled passwords has also been explored by Stock et al. [16]. They suggested that the password managers can prevent such attacks by using a placeholder dummy password for auto-filling and replacing it with the original one just before submitting the login form to the remote server. Blanchou et al. [17] describe several weaknesses of password manager browser extensions and implement a phishing attack that demonstrates the danger of automatic autofill. They suggest that password managers prevent the cross-domain submission of passwords. Unlike these previous work, this work have shown that an attacker can steal autofilled password by using specially crafted forms whose password fields don't appear in web page.

Conclusion:

This article has demonstrated that current built-in password managers of web browsers are vulnerable to special hidden field attacks targeting the stored passwords. This article has identified the root cause of the problem, namely the fact that password managers automatically fill out hidden password field with the clear-text password which is subsequently sent to the third party. In order to solve this problem, Browser Autofill Control System is proposed. Since the proposed system is developed to be an interface between users and web browsers, users can continuously use their web browsers as usual. Based on login page analysis, the system displays all input textboxes (including hidden textboxes) in the login page so that the user can decide whether or not to allow the browser to autofill password and other values to these textboxes. In order to abbreviate user confirmation on the login page which auto-filling has already been permitted, the system stores hash values computed from the login page and its CSS file. These values are used to verify whether there exists any update in the login page and CSS file accessed by the user next time. If there is no update in the login page and its CSS file, the system will automatically permit the autofill on the login page.

References:

[1] B. Ives, K. R. Walsh, and H. Schneider, "The domino effect of password reuse," Communications. ACM Volume 47, Issue 4, pp. 75-78, 2004.

[2] M. L. Mazurek, S. Komanduri, T. Vidas, L. Bauer, N. Christin, L. F. Cranor, P. G. Kelley, R. Shay, and B. Ur, "Measuring password guess ability for an entire university, " In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security (CCS '13). ACM, New York, NY, USA, pp. 173-186, 2013.

[3] R. Dhamija, J. Tygar, and M. Hearst, "Why Phishing Works, " In SIGCHI conference on Human Factors in computing systems, New York, USA, 2006. ACM.

[4] B. Englert and P. Shah, "On the Design and Implementation of a secure Online Password Vault," In ICHIT 09. ACM Press, 2009.

[5] B. Stock and M. Johns, "Protecting users against XSS-based password manager abuse, " In Proceedings of the 9th ACM symposium on Information, computer and communications security (ASIA CCS '14), pp. 183-194, 2014.

[6] D. Silver, S. Jana, D. Boneh, E. Chen and C. Jackson, "Password Managers: Attacks and Defenses," in Proceeding of 23rd USENIX Security Symposium, pp. 449-464, 2014.

[7] S. Gibbs, “Browser autofill used to steal personal details in new phishing attack”, https://www .theguardian.com /technology/2017/jan/10/browser-autofill-used-to-steal-personal-details-in-new-phising-attack-chrome-safari, posted 2017/1/10.

[8] Browser Autofill Phishing, https://anttiviljami .github.io/browser-autofill-phishing/, seen 2019/2/19.

[9] P. Rogaway, T. Shrimpton, "Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance," in Proceedings of Fast Software Encryption(FSE 2004), Lecture Notes in Computer Science, Vol. 3017, Springer-Verlag, pp. 371-388, 2004.

[10] A. W. Appel. Verification of a Cryptographic Primitive: SHA-256. ACM Trans. Program. Lang. Syst. Vol. 37, Issue 2, Article 7, 2015.

[11] Microsoft, "How to use the Windows Script Host to read, write, and delete registry keys", https://support .microsoft.com/en-ph/help/244675/how-to-use-the-windows-script- host-to-read-write-and-delete-registry-k, seen 2019/2/18.

[12] Mozilla, "How to permanently disable the save password feature in firefox?", https://support .mozilla.org /ja/questions/1158069, seen 2019/2/18.

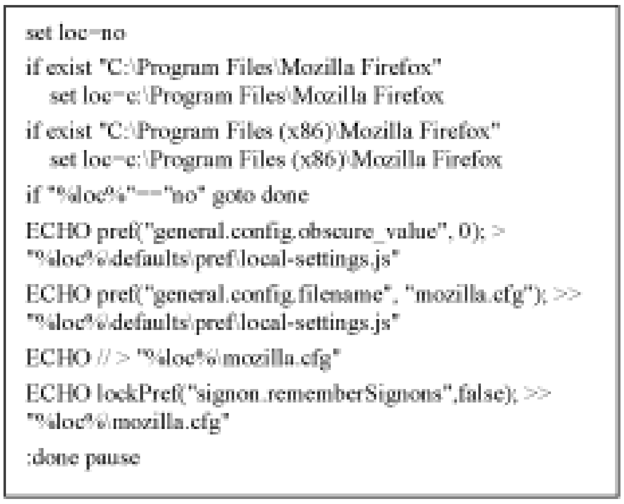
[13] S. Chiasson, P. C. Van Oorschot, and R. Biddle, "A usability study and critique of two password managers," In 15th USENIX Security Symposium (2006), pp. 1-16.

[14] R. Zhao, and C. Yue, "All your browser-saved passwords could belong to us: A security analysis and a cloud-based new design," In Proceedings of the third ACM conference on Data and application security and privacy (2013), ACM, pp. 333-340.

[15] Z. E. Ye, S. Smith, and D. Anthony, "Trusted paths for browsers," in ACM Transactions on Information and System Security (TISSEC) 8, 2 (2005), pp.153-186.

[16] B. Stock and M. Johns, "Protecting Users Against XSS based Password Manager Abuse," In Proceedings of the 9th ACM symposium on Information, computer and communications security, pp. 183-194, 2014.

[17] M. Blanchou and P. Youn. Password managers: Exposing passwords everywhere, https://isecpartners .github.io /whitepapers/passwords/2013/11/05/ Browser-Extension-Password-Managers.html.



```
set loc=no
if exist "C:\Program Files\Mozilla Firefox"
  set loc=c:\Program Files\Mozilla Firefox
if exist "C:\Program Files (x86)\Mozilla Firefox"
  set loc=c:\Program Files (x86)\Mozilla Firefox
if "%loc%"=="no" goto done
ECHO pref("general.config.obscure_value", 0); >
"%loc%\defaults\pref\local-settings.js"
ECHO pref("general.config.filename", "mozilla.cfg"); >>
"%loc%\defaults\pref\local-settings.js"
ECHO // > "%loc%\mozilla.cfg"
ECHO lockPref("signon.rememberSignons",false); >>
"%loc%\mozilla.cfg"
:done pause
```

Fig 9. An example of scripts that disables autocomplete feature of Mozilla Firefox.