FieldSecured: A Chrome extension for Autofill phishing prevention

By Fahad Alarefi, Computer Science Dept.
University of Oregon, Eugene, OR, USA

***Abstract*—This report analyzes the possible ways to prevent recently discovered design vulnerability in the Chrome browser. After analyzing the methods of prevention, it had been concluded that the best way for protection is to use a Chrome extension. Therefore, an extension called FieldSecured was developed and then published to Google Web Store. The extension's details of design and implementation are discussed in this report.**

Keywords—chrome, browser security, autofill, phishing, form phishing

I. INTRODUCTION

Internet access is continuously increasing over the world; people with different backgrounds and education are getting access to that network. Attackers interests come with the increase of Internet users; therefore the security of the Internet should be an interest of government and policy makers. Because using the audience little knowledge about technology, attackers can steal important information.

A technique that attackers use to get victims data is phishing. Which is about creating webpages that are identical to trustworthy websites and request sensitive information such as username and password. For example, a phisher might create a single webpage that it exactly similar to a us-bank and send scam mail congratulating the victim of winning a lottery and requesting them to login to their bank account to redeem their prize. From that point the attacker will receive numerous amount of login attempts from the victims and there is a high potential that most of login info received are correct credential.

Autofill is a feature that was introduced in 2011 to make the process of filling forms easier and more accurate [1]. Chrome desktop and smartphone users have the option to turn it off, however, it's turned on by default. The feature provides a faster way for filling forms. By a one click, it will fill all input fields it can recognize, despite the possibility that some of the fields are unseen. Google has found that with feature, users complete forms up to 30% faster [1].

Autofill is the most popular desktop browser, with 64% market [8], and this feature is turned by default. Therefore, many users are considered viable victims. And Autofill phishing might pose a great danger to that user base. This paper will analyze the security flaw, and discuss the proposed solution, FieldSecured, which is a browser extension that solves the problem adequately.

II. PROBLEM

A. *Origin of the Problem*

Chrome Autofill form phishing is a design flaw in the desktop and mobile browsers. By requesting normal data, like username and password, the attacker can receive much more information than what the user thinks. Resulting in leaked personal information without user's consent. However, there are no current incidents from any websites.

The design flaw was recently pointed out, in January 2017, by the Finnish web developer Viljami Kuosmanen [12]. Since than, many news outlets had exposed the problem to the public, including The Guardian [10], The Register, and Lifehacker [11]. Also, developers have been posting it in the browser developers community since a long time.

The problem was first discovered in 2012 through Issue 132135 of the Chromium bugs report forum. The author suggested that Chrome should not fill those "hidden" fields but leave them empty. In 2017, a project team has acknowledged that they are working on it but have no solutions yet [3]. It has been more than 5 years and the problem is not solved from Google.

B. *How it Can be Exploited*

As it pointed out by a Google engineer, Chrome doesn't respect developer's option of opting out of the Autofill. Ido Green states that web developers have the option to turn off the autocomplete feature, by adding *autocomplete="off"*, however,

Google chrome will respect that tag for autocomplete functionality but will overpass that option to apply forms Autofill [1].

A form field, or form control, is used to submit data from the user's browser, to the server. And an input field is a type of form fields that has many kinds, like text, checkbox, and email.

Autofill supports many kinds of input fields including select, checkbox, and text. The way it works is by matching the input field's name-tag against the user's cloud-based "addresses" database. When a matching address is found, the browser will show a list, see Fig. 1, of all matching addresses that it found. If the user clicked one of the list-items than the browser will Autofill the requested field in addition to any field in the form that has a matching name-tag.

For example, if the user is filling a form and started typing, or clicked the white space, in the email field, then browser will look for any email that starts with the typed text. If any found, then the browser will search the whole form looking for additional fields it can fill. When the user hovers an option from the list, the browser will instantly Autofill matched input fields. To distinguish it from user-filled ones, Autofill-filled fields are marked by a yellow background.

In case this vulnerability is exploited then the website will receive more information then what the user has filled. For instance, Fig. 2 shows the text fields that are shown to the user when they filled the form. In this case, the user can only see two fields, email and name. An ordinary user will not have doubt that there are more hidden fields. Yet in that same page, there are many more "hidden" fields that are shifted to the far left. As seen in Fig. 3, after the user submitted the form, the website received sensitive data like phone number and home address.

C. *Browser Processing Model*

WHATWG HTML5 standard has given more attention to the autocomplete field; the specs have a full section discussing the ways to implementing a standardized set of field-names (2). The initiative's goal is to make all browser's autocomplete act on similar way, to make it easier and more efficient for the developer. For instance, Autofill detail tokens were introduced to group a section of form's fields together so that the browser won't fill the fields altogether. By replacing * in "section-*" and putting it in all field's autocomplete tags, the browser will only auto fill those grouped inputs [2].

If an input field is a part of a form that the user is filling and the field is out of the human sight, then the browser will deal with it as if it was viewable. Since the browser processes the page by using DOM elements, then it doesn't recognize what is included in the user's viewport, and therefore, it will fill it with data. Reviewing the Chromium source code, it's unclear whether the fields status are checked against any filters or not.

III. RELATED WORK

A. *Journal Reviews*

In preparation for this work, I reviewed multiple works that evaluate the security of Chrome. In general, many security experts find that Chrome is one of the most secure browsers in the industry. A study done by Charles Reis, Adam Barth, and Carlos Pizano identified three techniques that any browser developers should use to help protect users from web malicious-attacks:

- Reducing the vulnerability severity: limit the damage by applying the least privilege in the browser's architecture.
- Reducing the window of vulnerability: provide a constant updates to users automatically.
- Reducing the frequency of exposure: by filtering out known malicious content.

The researchers found that the Google Chrome team has focused in each of these techniques to create a very secure browser environment [4].

B. *Current Solutions*

After looking for current solutions for this security issue, I found a published browser extension that was published by a developer called bramas.fr [5]. The implementation is too simple and doesn't provide enough security. The extension only shows two things: the total number of Autofill-ed fields and what content is being filled. As seen in Fig. 4, I think that this solution damages the user's overall experience because it adds a tooltip for every form that the user fills. Resulting in unstructured overall look and feel. In addition, it requires the user to count every field

they will be filling and matching it with the shown total so it changes the main goal of Autofill feature which ease of use.

C. *Autofill in Other Browsers*

The autofill feature is implemented differentially in Firefox and Safari. The two browser use a more secure way of dealing with Autofill. In Firefox, the user cannot auto fill all form's fields. They must go over each field and choose the suggested data. However, it's different than autocomplete; the browser smartly links the information provided to the same "address". On the other hand, Safari uses the most intuitive way of implementing Autofill. As in Fig. 5, the browser allows the user to see all information being filled in the form before it fills it. This way the browser can notice the information that should not be submitted.

IV. DESIGN AND IMPLEMENTATION

At first, I though that the best way to prevent such attacks is by disabling the Autofill feature altogether. However, I revised my idea after implementing FieldSecured because the browser extension doesn't cause any disruption to the user's regular use, yet it solves the problem nearly perfectly.

A. *Design Goals*

In order to achieve the best results, the solution has to guarantee three main criteria. If those goals are met in the final product, then it should serve the user the best experience. The goal are as follows:

- *User convenience*: In the first place, Autofill is used for convenience and ease of use. Therefore, the solution should provide the user the best browsing experience with no interruption or attention.
- *Low use of resources*: Since the extension will be working continuously in background, then the solution should not consume a lot of computing resources.
- *High accuracy of identifying phishing forms:* The solution should be able to figure out malicious forms with high accuracy. Without this objective, the implementation would be useless since web-phishers are continuously developing.

B. *Testing Methods*

In order to achieve the last goal, I needed to develop testing methods and scenarios. So first we will start with text field display status. There are three ways a field can get of the user sight: use *type="hidden"* in the field's input tag, use *display="none"* in the CSS properties, or use CSS tricks to hide the elements, like to position the fields to the far left so that it's out of the user's viewport. After testing the three ways, I founded that Chrome disable's Autofill for the first two methods. However, some variations of the third method can penetrate the browser and cause it to fill the data.

To test what data can be stolen from the user's addresses database, I used a form with all possible text fields from WHATWG's list of autocomplete types [6], and margined all fields, but first and last name, to -50000px. Then from Chrome setting, I added a test "address" that has personal information such as name, address, email, etc. After tasting the form with the newly added address, the webpages received more data than "first and last name", see Fig. 6. In addition, many other tricks can overcome Autofill such as adding *opacity: 0;* to the style. Or using the CSS property *visibility: hidden;* Also, *clip-path: polygon(0px 0px,0px 0px,0px 0px,0px 0px);*. All of these tricks will result in a "hidden" fields that take space in the page but not seen by the user [7]. I have tasted all above ways, and found that all of them can trick Autofill and cause it to fill the data.

C. *Ways of Detection*

My implementation will detect phishing forms by looking for three anomalies in the field's properties:

- Check whether the field is the user's viewport or not. By comparing the window's height and width against the field's position, it can be concluded that the field is within the browser's viewport or not. This eliminates the possible attacks of *margin: -50000px*; and *position: relative; top: -50000px.*
- Check whether the field has any CSS properties from theses: visibility, opacity, clip-path. And make sure that they are not used by to phish for data. For *opacity,* check if the value is less than 0.5 or not. I

chose that number arbitrary because I think less then that value might cause the user to not see it.

D. *Implementation*

Chrome browser extensions have two parts of code: a content script, and popup menu code. The content script is responsible for all the code that happens in the background. I have setup the extension to work in any page that matches ["http://*/*", "https://*/*", "file://*/*"]. So it will be working in all possible pages. Although the extension will get executed in every page visit, I have tried to achieve the second goal and use lowest possible use of resources.

The initial idea of implementation was to make the code work only when Autofill feature is used in any of the forms. However, I found that the browser doesn't dispatch any special JavaScript events when Autofill is used. On the other hand, Chrome adds the pseudo-CSS tag -webkit-autofill to any field that the feature had executed on. Therefore, the extension will use this tag to distinguish Autofill fields.

On any input change event, a code will trigger checking whether the field is filled by the user or not. If not, then it will check the visibility of the input by passing it to the isVisible function. The function will check the properties detailed in "*c. Ways of Detection*". If the field meets any of them, then it will add it to an array of potential fields.

As simply explained in Fig. 7, when the user resizes the window or scrolls down, the code will trigger and reevaluate all fields in the potential fields list. This way, it eliminates fields that were filled by Autofill and located down in the page (initially unseen by user). So for any Autofill field, the user has to see the field before it gets submitted.

Finally, when the user submits the form, the extension will check out the form and make sure that it has no unseen Autofill filled fields. In case there was, then the form will not be submitted and a message of action will be shown to the user. As demonstrated in Fig. 8, an attentive message will be shown in red, and the hidden submitted data will be printed. The user has two options: report the website to Google as a malicious website or continue anyway and submit the data. The first option will redirect the user to a malicious software report page maintained by Google. For the reporting website, another alternative was a website called stopbadware.org, however, Google's website looked more genuine.

V. RESULTS AND ANALYSIS

A. *Final Implementation Result*

The practicability result of my implementation is limited to what I created as a test. However, the result of the extension as a product is an adequately aesthetic product that has practical functionality. To the user, as a proof of functionality, the extension will have a simple popup small page that has a counter of the total times that the extension has found a phishing fields, demonstrated in Fig. 9.

B. *Analysis*

In order to come up with better practical results, I researched what are the possible ways to hide an element in CSS. I then used the findings to better tune the ways my extension can find a "hidden" field. The result is a Chrome extension that works continuously in the background and meets all previously defined goals.

In detail, FieldSecured won't intervene to the user's regular use. Other then extension's logo shown in the extensions bar, the user won't even notice that something is working in the background. It won't print any text in the page unless a phishing suspicion has happened.

Also, the extension uses the lowest resources possible. It is in idle state until Autofill use had happened. So the computation calculations and checkups won't be executed until Autofill is used.

In addition, by checking the potential field against three different techniques of hiding the field guarantees a high accuracy of identifying phishing fields.

VI. DISCUSSION OF COMPLICATED ISSUES

The complexity of web development makes the goal of perfectly fixing this issue impossible. At the end, there must be ways to overcome my implementation. In addition, it is interesting that there are no ways to know whether an html element is seen to the human eye or not. I feel that such function should be provided in JavaScript or

at least, jQuery because it helps web developers in better designing their pages.

In addition, one issue that makes the extension not very useful is that these types of phishing attacks are not common, and barely ever happened. Because most users use Autofill either for information about shipping, or billing, or using it to fill personal information for signup. Both of these uses require a trusted website owner. Because the user won't buy/signup in the first place if the website is not trusted.

VII. Conclusions and Lessons Learned

The popularity of Chrome browser makes it the attention of many hackers. The Autofill forms vulnerability is a basic UI flaw that should had been fixed a long time a go, yet, the team behind Chrome has not offered any solutions. Therefore, I have created a browser extension for Chrome desktop that attempts to solve the issue. By following three goals: user convenience, low use of resource, and high identification accuracy, the extension resulted in effective approach of solving the problem. FieldSecured will be published to the extensions market for free so that users get protected from this type of attack.

Many lessons were learned throughout the implementation of this project. Especially, the architectures of browsers and how do they render pages. In addition, jQuery was used mainly to make the process of creating the extension easier and faster. However, I think that it's possible to achieve similar results using JavaScript only.

VIII. Figures and References

A. *Figures*

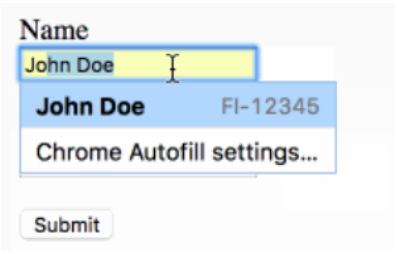

Figure 1 Shows how the list of addresses are shown [12]

Figure 2 Shows the fields shown to the user [12]

Figure 3 Shows what information has the website received [12]

Figure 4 demonstrate bramas.fr's Autofill Checker [5]

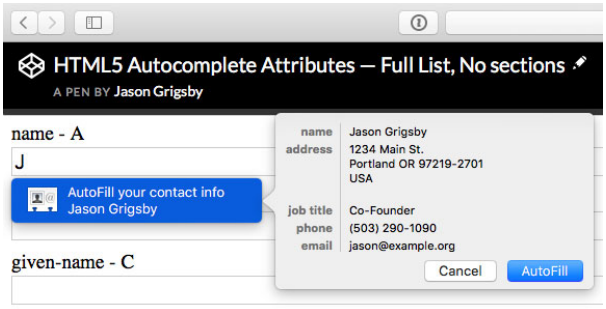

Figure 5 Safari's implementation of Autofill [9]

```
[given-name] => Test
[family-name] => Name
[honorific-suffix] =>
[honorific-prefix] =>
[additional-name] =>
[additional-name-initial] =>
[nickname] =>
[street-address] => 1477 E 13th Ave
[address-line1] => 1477 E 13th Ave
[address-line2] =>
[address-line3] =>
[locality] => Eugene
[city] =>
[administrative-area] =>
[state] =>
[province] =>
[region] => Oregon
[postal-code] => 97403
[country-name] => United States
[email] => test@email.com
[tel] => 5410001111
```

Figure 6 The results when form is submitted

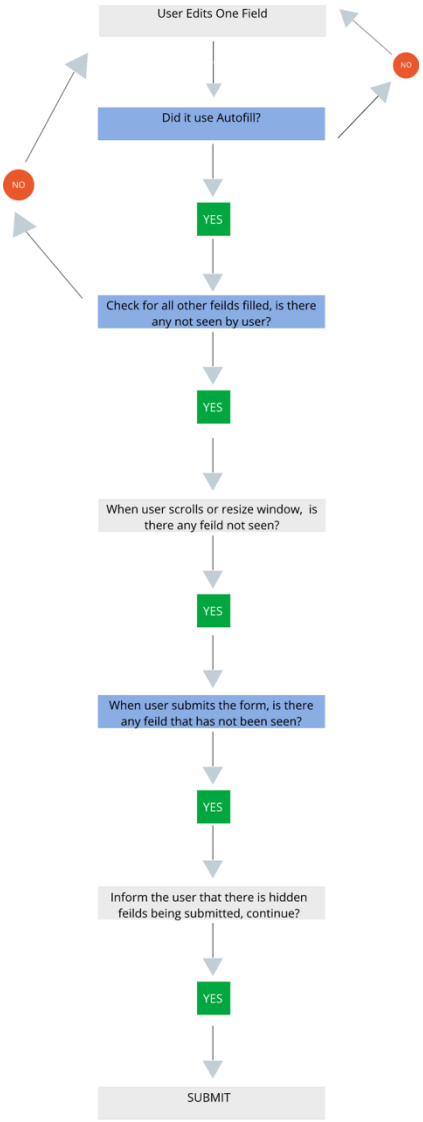

Figure 7 Flowchart shows how FieldSecured works

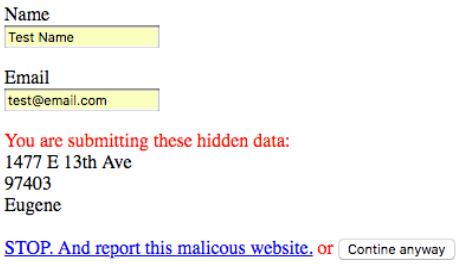

Figure 8 What FieldSecured shows when there is a suspicion of phishing

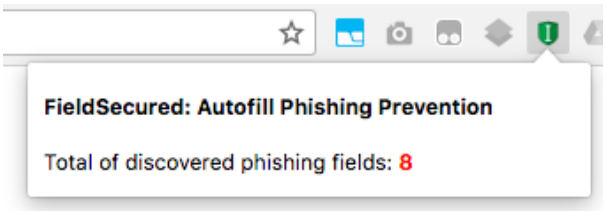

Figure 9 How FieldSecured is shown to the user

B. *References*

[1] Green, I. (2015, June). Help users checkout faster with Autofill | Web | Google Developers. Retrieved March 18, 2018, from https://developers.google.com/web/updates/2015/06/checkout-faster-with-autofill?hl=en

[2] WHATWG. (2018, March 16). HTML STANDARD. Retrieved March 18, 2018, from https://html.spec.whatwg.org/multipage/form-control-infrastructure.html#autofill

[3] Chrome's Autofill feature circumvents anti-spam honeypot hidden form field techniques. (2012, June 11). Retrieved March 18, 2018, from https://bugs.chromium.org/p/chromium/issues/detail?id=132135

[4] REIS, C., BARTH, A., & PIZANO, C. (2009). Browser Security: Lessons from Google Chrome. Communications Of The ACM, 52(8), 45-49.

[5] Bramas.fr. (n.d.). Autofill Checker. Retrieved March 18, 2018, from https://chrome.google.com/webstore/detail/autofill-checker/nfdenjdodgbjbcbocechcbkhncakpieb

[6] WHATWG. (n.d.). Autocomplete Types. Retrieved March 18, 2018, from https://wiki.whatwg.org/wiki/Autocomplete_Types

[7] Rathi, B. (2016, July 23). Five Ways to Hide Elements in CSS. Retrieved March 18, 2018, from https://www.sitepoint.com/five-ways-to-hide-elements-in-css/

[8] StatCounter Global Stats. (2018, February). Desktop Browser Market Share Worldwide. Retrieved March 18, 2018, from http://gs.statcounter.com/browser-market-share/desktop/worldwide

[9] Grigsby, J. (2017, September 12). Autofill: What web devs should know, but don't. Retrieved March 18, 2018, from https://cloudfour.com/thinks/autofill-what-web-devs-should-know-but-dont/

[10] Gibbs, S. (2017, January 10). Browser autofill used to steal personal details in new phishing attack. Retrieved February 13, 2018, from https://www.theguardian.com/technology/2017/jan/10/browser-autofill-used-to-steal-personal-details-in-new-phising-attack-chrome-safari

[11] Allan, P. (2017, January 11). Your Browsers Autofill Data Can Be Phished, Heres How to Stay Safe. Retrieved February 13, 2018, from https://lifehacker.com/your-browsers-autofill-data-can-be-phished-heres-how-t-1791084371

[12] Kuosmanen, V. (2017, January 07). Browser-autofill-phishing. Retrieved February 13, 2018, from https://github.com/anttiviljami/browser-autofill-phishing