AUTOFILL SECURITY

PROJECT REPORT

Pawan Goyal pawan14@mit.edu

Richard Liu rtliu@mit.edu

Scott Perry seperry@mit.edu

Ian Limarta limarta@mit.edu

May 22, 2021

ABSTRACT

Autofill is used in our day to day life but has sever security threats. Previous work has shown that malicious users can create forms with hidden input fields that are able to surreptitiously gather information from the user that the user did not want to provide. This paper will further investigate the vulnerabilities in autofill and provide prototypes for potential remedies for these vulnerabilities. First, we provide a security policy to clearly specify what secure autofill functionality would look like. Then we demonstrate four new methods for finding input fields using CSS and HTML functionality to show that there are still many undiscovered vulnerabilities. Finally, we build a BookMarklet that is able to identity some hidden input fields so that the user can avoid having their information stolen.

1 Introduction

Today, many users take advantage of the convenience of their browser's autofill feature when filling out all sorts of web forms on the internet. Typing out your full name, or full address, can be tedious; but the cost of autofill's convenience can be great (Lin et al. [2020]). In order for autofill to be secure the user needs to be aware of exactly what information they are providing to the web-form creator. However, it is possible for a malicious actor to hide what information they are collecting in the webform during autofill by hiding input fields. Therefore, a user who just wanted to provide their name could end up giving their birthday or address as well.

Though previous work by Lin et. al. has investigated some of the vulnerabilities in autofill this paper will extend on that work in three ways. First, we will provide a security policy that outlines exactly what secure autofill functionality would look like. Then we demonstrate additional attacks that show that there are even more vulnerabilities in autofill then the Lin et. al. found. Finally, we will present a BookMarklet that partially protects against these attacks by scanning for input fields hidden in a variety of ways.

2 Background and Previous Work

While there has been much work in similar security-fill features such as password managers, there has not been extensive researching looking into the autofill feature. A recent paper by Lin was the first extensive analysis of autofill over many different brewers where they demonstrated potential exploits with autofill, different attacks by manipulating visual elements, and results of a crawling experiment designed to search for hidden autofill elements in the top $100,000$ websites. In addition to proposing different visual attacks, they also introduce new side-channel attacks using menu bars for extracting information even when users opt not to use autofill (the user must of course enable autofill). It is not surprising that many websites have incorporated autofill to convenience the user but their search remarkably shows that many of the most highly-trafficked websites do contain hidden elements. Figure $1a$ is a quote of their results and Figure $3b$ lists what attacks they found for different browsers (Lin et al. [2020]).

	Sites w/ Autofill	Sites w/ Hidden Field	Pages w/ Autofill Forms	Pages w/ Hidden Autofill
Firefox	21,589	5,295	92,063	8,760
Chrome	31,621	1,843	83,054	2,776

(a) Results of Lin et al.'s web crawler over the top 100K websites

Techniques	Firefox	Chrome	Brave	Edge	Safari	Opera
CSS Display	✓	✗	✗	✗	✗	✗
CSS Visibility	✓	✗	✗	✗	✗	✗
CSS Opacity	✓	✓	✓	✓	✓	✓
Covered by overlay	✓	✓	✓	✓	✓	✓
Non-effective size	✓	✓	✓	✓	✓	✓
Off-screen placement	✓	✓	✓	✓	✓	✓
Ancestor's overflow	✓	✓	✓	✓	✓	✓

(b) A list of different attacks on several popular browsers

Figure 1: A figure with two subfigures

3 Security Policy

We propose a security policy for browser's autofill features. We explore the negative security consequences of current implementations, and the recommendations we give in the follow security policy outline a secure and private method of implementing the autofill feature.

3.1 Definitions

Personal Information Set A personal information identifier set describes a set of personal identifying information belonging to some person. This information can include a name, organization, street address, phone, and email. Web browsers allow users to add and remove different sets of personal information for different context. For example, a user may have a set of information describing their name and address but have a different set which includes credit card information.

Autocomplete/Autofill Web browsers offer an autofill feature, so that when filling out web forms, a user can either choose to type in the necessary personal information, or instead use the autofill feature. This features lets all fields be automatically filled in with a click, according to the user's stored personal information set.

Web Forms Web forms consist of collections of input fields which ask for a user's personal details such as their name or email. A malicious web form might transparently asks a user for some set of personal details (requesting only a name and email), and at the same time secretly requests from the browser additional personal details (a mobile number).

3.2 Users and Their Roles

Web User

The user interacts with the webpage through a web browser. He may click, copy or submit data on the web page but we assume he is not malicious so he cannot change the source code of the HTML web page under consideration.

He reserves the right to use optional functionalities provided by the web browser like autofill. He might use autofill to automatically fill the fields or may choose to manually fill each of them.

He also could add additional functionalities to the web browser like bookmarklets or extensions but cannot directly interfere with the web browser's source code.

Web Browser

Web Browser is responsible to implement autofill feature. It securely stores the users' data and provides the utility to automatically fill them in the fields marked with "Auto-complete".

However, web browser doesn't and shouldn't hold the power to force the user to use autofill. It can also not automatically fill the fields marked as auto-complete without the user's direct consent/click.

Web Page Creator

The web-page creator is responsible to for all the content on the web-page. He is responsible to maintain the page and fix it if anything breaks in the HTML code.

He also reserves the right to use/store the data submitted by the user through his web-page, until previously agreed upon otherwise.

Assumptions

Throughout our analysis, we assume web browser is the honest party who doesn't try to steal any data from the user. We also assume that data is stored securely on the browser and it is computationally infeasible to steal it directly.

Besides, we assume that all communication channels are secure and the attackers cannot extract any meaningful information by eavesdropping on the network. The only way attackers could interact with the users is through malicious web-pages and all other pathways are either secure or blocked.

3.3 Privacy

We consider a privacy-satisfying autofill implementation as one which informs the user exactly what personal details are being revealed to a website, when autofill is used. We consider a privacy-violating autofill implementation as one which unexpectedly reveals some personal details to a website, without the user's knowledge. Here, we outline two privacy-satisfying options of implementing an autofill feature.

Simple Autofill When a user chooses to use the autofill feature, the browser should present to the user a clear summary of all of the details that will be filled in.

Advanced Autofill When a user chooses to use the autofill feature, the browser should present to the user an overview of their stored personal details. Here, the user can precisely select those personal details that they wish to populated the web form with.

4 New Attacks

The paper by Lin et. al. explored several different potential attacks. Many of these attacks were very effective at hiding input fields so finding new attacks may seem unnecessary as attackers could just exploit the published vulnerabilities. However, it is likely that the creators of the browsers will patch these vulnerabilities over time forcing attackers to utilize different methods to hide input fields if so desired. Therefore, we outline several new potential attacks below so that these vulnerabilities can be patched preemptively.

We introduced four new potential vulnerabilities - three based on CSS functionality and one based on HTML functionality. The CSS functionalities used were CSS Animations, CSS Opacity Filters, and CSS Filter Functions. We find that each of the attacks based on these functionalities work on all tested browsers: Firefox, Chrome, and Edge. The HTML functionality used was the HTML Global Attribute. This attack based on this functionality was less effective only working against the Firefox browser and not against the Chrome and Edge browsers.

4.1 CSS Animations

The attack based on CSS Animations works similarly to the attack based on CSS Overlays described in Lin et al. The simple Keyframe animation consists of a white box that appears over the input fields that the attacker wishes to hide for a set duration. By setting this duration to a very large value the animation is extremely unlikely to finish revealing the input fields in the time that the user is on the page.

```css
div {
  width: 300px;
  height: 195px;
  background-color: white;
  animation-name: example;
  animation-duration: 30000s;
  position: relative;
}

@keyframes example {
  from {background-color:white; left:0px; top:-80px;}
  to {background-color: white;}
}
```

Listing 1: Setup Code for Attack based on CSS Animations

This attack based on CSS Animations has a few advantages over the attack based on CSS Overlays described in Lin et. al. The first is that it may be harder for the browser to block because animations can have a legitimate uses like showing a privacy alert. Furthermore, if movement is added to the animation it could be hard for the browser to determine whether or not the animation is covering an input field and is therefore likely to be malicious and so should be removed.

4.2 CSS Filter Function

The filter CSS property applies graphical effects like blur or color shift to an element. Filters are commonly used to adjust the rendering of images, backgrounds, and borders. Besides the CSS attributes discussed above, we could also use these CSS filter functions to maliciously extract sensitive information from the users. We now briefly discuss various ways to use them to create hidden fields.

4.2.1 CSS Opacity Filter Function

Distinct from `opacity` CSS property, the `opacity()` CSS filter function can, in a similar way, be used to set the transparency of page elements, including input fields. Listing 2 provides an example class which makes the input filed hidden. We find that this attack works in all tested browsers, including Firefox, Chrome, Edge.

```
<style>
    .blurry {
      /*filter: blur(1000px);*/
      /*filter: brightness(1000%);*/
      filter: opacity(0);
    }
</style>
```

Listing 2: Code to use Opacity Filter to create hidden Fields

4.2.2 Other CSS Filter Functions

CSS filter functions, as mentioned above, represent plethora of graphical effects that can be applied to images, general page elements, and even input fields. This includes effects like `grayscale()` and `saturate()`, but of particular interest are those that have usage in hiding input fields. Each of the following CSS filter functions, with the suggested parameters, can hide an input field.

```
blur(1000px)
brightness(1000%)
contrast(0)
invert(0.5)
```

This gives four attacks, each distinct in flavor. We find that these attacks work in all tested browsers, including Firefox, Chrome, Edge.

4.3 HTML Global Attribute

HTML elements offer several global attributes, and of particular interest is the `hidden` attribute which can be used to hide page elements, including input fields. An example attack can been seen in Listing 3. We find that this attack works in Firefox, and does not work in Chrome, Edge.

```
<div id="drop-down" hidden>
<!-- <div id="drop-down" style="display: none;"> -->
    Address<input id="address" autocomplete="address">
    Zipcode<input id="zipcode" autocomplete="zipcode">
    Phone Number<input id="phone" autocomplete="phone">
    Organization<input id="organization" autocomplete="organization">
    E-mail<input id="email" autocomplete="email">

</div>
```

Listing 3: Code to use Global Hidden attribute to create hidden Fields

5 100k Alexa websites

We analyze the top websites, as instances of this autofill exploit in the wild can inform our process. While a previous study revealed several instances of maliciously hidden input fields in the wild (Lin et al. [2020]), we are interested in reproducing and extending those results.

In particular, our program searches for usage of the `"opacity: 0"` input fields across the top 2000 websites. Our python script loads all websites of interest using Selenium and chromedriver, and executes Javascript to determine which input fields were programatically hidden (4).

In our analysis of the results, we find that several websites did use hidden input fields not for malicious purposes, but rather for styling. For example, one website would hide their input fields until a user initiated the registration process upon which the input fields would be revealed. Ultimately, among the top 2000 websites we do not find any malicious use of hidden input fields.

6 BookMarklet

So far, we have seen that the autofill feature is prone to plethora of attacks and can be successfully used by attacked to extract user's sensitive information without their knowledge. However, in this section we present bookmarklet which could successfully identify these threats and thus could be used to remedy previously mentioned attacks.
A bookmarklet is a small software application (generally JavaScript) stored as a bookmark in a web browser, which typically allows a user to interact with the currently loaded web page in some way. Besides, it is far quicker to develop and test as compared to a browser extension and thus justifies our choice. We developed our bookmarklet to detect three major attacks : **Zero Opacity**, **Zero Size** and **Offset relative to Main screen** and each of them are briefly discussed below.

6.1 Zero Opacity

The bookmarklet scans the loaded web page and look for all instances of **input** field with autofill enabled. It then checks for its opacity and alerts if it resolves out to be zero. However, finding the real opacity of a HTML object is tricky. Even if the input field has opacity 1 but is under the parent with opacity 0, it would not be visible to the user on the loaded web page since elements may not be more opaque than their parents. Moreover, opacity is not inherited from the parent which further exacerbates the situation. So to find the actual visible opacity, one needs to multiply object of interest's opacity with that of its parents. The relevant code snippet which performs this calculation can be found at Listing 4:

```
function hierarchicalOpacity(element) {
    let opacity = 1;
    while (element && element.nodeType == 1) {
        const op = getComputedStyle(element).opacity;
        if (op) {
            opacity *= parseFloat(op);
        }
        element = element.parentNode;
    }
    return opacity;
}
```

Listing 4: Code to find true opacity of HTML object

So once the true opacity is evaluated, we check if its close to zero. If yes, we hierarchically make the opacity of the object under inspection 1 and highlight it with red to make it visible to the user. Our bookmarklet in action could be seen in Fig 2

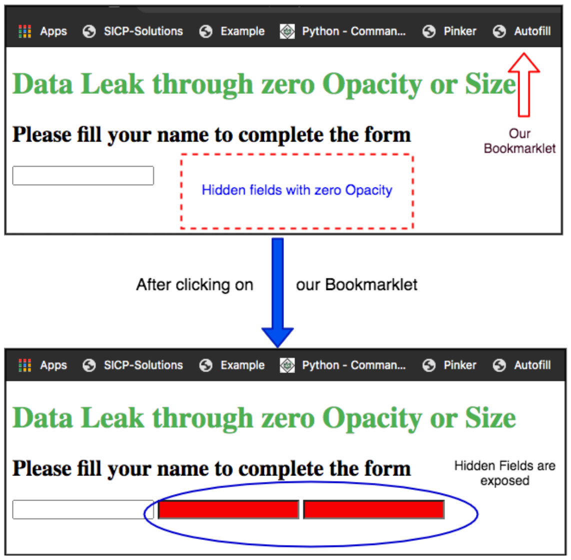

Apps
SICP-Solutions
Example
Python - Comman...
Pinker
Autofill
Data Leak through zero Opacity or Size
Please fill your name to complete the form
Our Bookmarklet
Hidden fields with zero Opacity
After clicking on
our Bookmarklet
Hidden Fields are exposed

Figure 2: Zero Opacity Attack

6.2 Zero Size

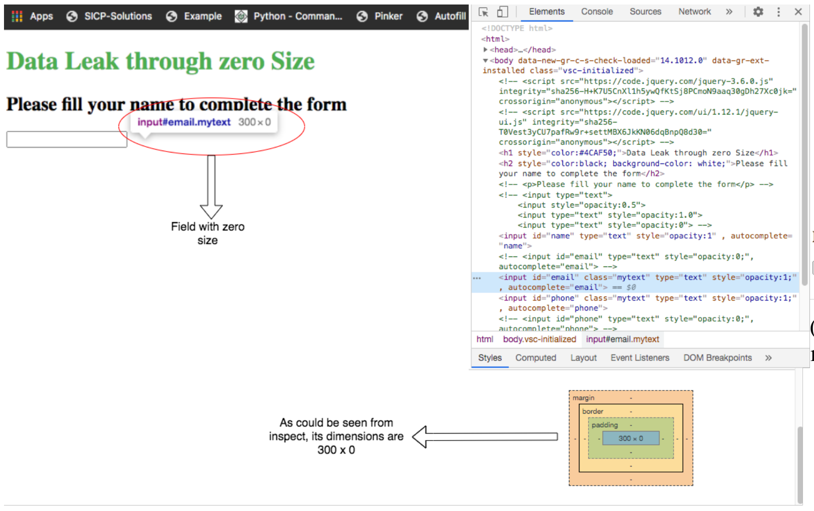
As in the previous case, bookmarklet again filters out the input boxes with autofill activated. Then it checks for the size, which includes checking if width or height is zero. If it finds any such instance, it changes the dimension of the hidden input box to non zero height and width and also colors it green to notify the users (see Listing 5). The bookmarklet for zero size hidden fields can been visualized through Fig 3.

```
function showSizeAttackedElem(sizeAttackedElements) {
    var style = document.createElement('style');
    style.innerHTML = `
    .normalSize {
        width: 145px;
        height: 15px;
        padding: 2px;
        border: 2px;
        margin: 2px;
        border-style: inset;
        border-width: 2px;
        border-color: #767676;
        background-color: #4CAF50;
      }
    `;
    document.head.appendChild(style);
    for (i = 0; i < sizeAttackedElements.length; i++){
        sizeAttackedElements[i].className = "normalSize";
    }
}
```

Listing 5: Code to show zero sized input fields

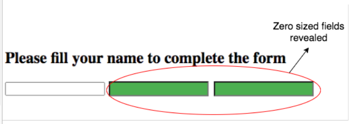
6.3 Offset Elements

The *position* attribute of CSS allows elements to be placed based on absolute coordinates or relative positions from other elements. If used improperly, an adversary may place the element to the extreme left or right of the



```
<!DOCTYPE html>
<html>
▸<head>…</head>
▾<body data-new-gr-c-s-check-loaded="14.1012.0" data-gr-ext-
installed class="vsc-initialized">
    <!-- <script src="https://code.jquery.com/jquery-3.6.0.js"
    integrity="sha256-H+K7U5CnXl1h5ywQfKtSj8PCmoN9aaq30gDh27Xc0jk="
    crossorigin="anonymous"></script> -->
    <!-- <script src="https://code.jquery.com/ui/1.12.1/jquery-
    ui.js" integrity="sha256-
    T0Vest3yCU7pafRw9r+settMBX6JkKN06dqBnpQ8d30="
    crossorigin="anonymous"></script> -->
    <h1 style="color:#4CAF50;">Data Leak through zero Size</h1>
    <h2 style="color:black; background-color: white;">Please fill
    your name to complete the form</h2>
    <!-- <p>Please fill your name to complete the form</p> -->
    <!-- <input type="text">
        <input style="opacity:0.5">
        <input type="text" style="opacity:1.0">
        <input type="text" style="opacity:0"> -->
    <input id="name" type="text" style="opacity:1" , autocomplete=
    "name">
    <!-- <input id="email" type="text" style="opacity:0;",
    autocomplete="email"> -->
... <input id="email" class="mytext" type="text" style="opacity:1;"
    , autocomplete="email"> == $0
    <input id="phone" class="mytext" type="text" style="opacity:1;"
    , autocomplete="phone">
    <!-- <input id="phone" type="text" style="opacity:0;",
    autocomplete="phone"> -->
```

(a) Web page before using our bookmarklet


Zero sized fields revealed
Please fill your name to complete the form

(b) Web page after using our bookmarklet.

Figure 3: Zero Size attacks

screen when the user does not have permission to scroll horizontally. One way an adversary can push elements is to disable the *overflow* attribute of an element to restrict horizontal scrolling and place the autofill textbox within this element. Figure 4 shows a partially displayed textbox. Unlike the opacity attack which must use the explicit *opacity* attribute to hide fields, offset attacks can use positioning in complicated ways so that it is hard to compute whether an element is truly outside of the window.

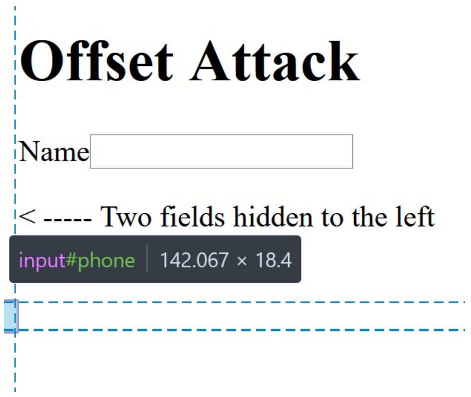

Offset Attack
Name
< ----- Two fields hidden to the left
input#phone 142.067 × 18.4

Figure 4: A website with one visible element and two hidden elements by offset. The element highlighted in blue shows one of the elements partially hidden.

In our experimentation, we found that certain configurations of position can hide elements partially on screen and yet remain totally visible. A standard CSS element consists of four layers which creates the total element: the body, padding, border, and margin which all together determine the position of the element. It is unclear why exactly this pathological example occurs but this exemplifies the many variations of offset. In our bookmarklet, we have used the `getBoundingClientRect` for an element which returns absolute coordinates of an element on the screen. We then checked whether the x-coordinate was negative or larger than the width of the window (Listing 6).

One limitation is that this technique clearly does not find all offset attacks as hinted from the pathological example since the coordinates may be within the viewport. Furthermore, an advesary may choose to only partially hide the element and therefore still have coordinates within the viewport. A proper detection algorithm would incorporate the dimensions of the different layers and compute coordinates of nested elements in an iterative manner.

```
function determine_offset(element) {
    var rect = element.getBoundingClientRect();
    return rect.right <= 0 || rect.right > window.innerWidth;
}
```

Listing 6: Code to find offset

7 Future Work

In the past few weeks, we explored various autofill security threats across different browsers. We also successfully developed a bookmarklet which would help the users to detect these malicious web-pages and alert them about these hidden fields. However, still lot of areas remain untouched or unanswered and given more time and resources in future, we would like to build upon our current work and achieve following goals:

- To start with, we would like to explore more security threats associated with autofill. We speculate that there might be more ways to hide fields from the users and in the future research, we would like to tabulate the complete exhaustive list of such attacks.
- We would also like to expand our analysis of the prevalence of such input field-hiding attacks in the wild. Possible extensions include expanding the search to the full, top 100k websites, and to include links traversed from a parent page, as described in Lin et al. [2020]. A future iteration of this script might also include checks for, not just opacity attacks, but the full suite of input field hiding methods.
- Currently, our bookmarklet detects three major attacks: zero opacity, zero size and offset relative to main screen. In future, we would like to add all the attacks found so far and also the new attacks discovered along the way.
- Besides, our proposed bookmarklet uses color coding to differentiate between different attacks. For example, red for zero opacity, green for zero size and yellow for offset. However, as we would add more and more attacks in the future, it would become perplexing for the user to remember these color coding. Therefore, we hope to come up with better ways to alert the users about the different hidden fields on the web-page beside our current color scheme.
- Another shortcoming with our bookmarklet is that it demands the user to explicitly click on it to detect the hidden fields. However, the user might forget to check for the attack and this would defeat the purpose of the bookmarklet. Therefore, in future, we hope to convert out current bookmarklet into an chrome extension that automatically runs the relevant JavaScript code whenever a new web-page is loaded. We also hope to extend it further to work across various popular web browsers, and not just Google Chrome.
- Finally, we aim to use our final product against real websites and not just our simulated test cases. That would provide a real test to our proposed extension and if everything works our as anticipated, we look forward to open source our project so that other people could build upon this.

8 Conclusion

We have developed several proof-of-concept malicious webforms that hide input fields to unexpectedly acquire user personal information. We find that creating such autofill exploits is not difficult, and recommend both web browsers and users to take appropriate action; web browser might better inform users about autofill's behavior, and users might disable the autofill feature. Despite the convenience afforded by autofill, we hope that users will stay vigilant about usage of their personal information on the internet.

References

Xu Lin, Panagiotis Ilia, and Jason Polakis. Fill in the blanks: Empirical analysis of the privacy threats of browser form autofill. *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020.