NoTabNab: Protection Against The "Tabnabbing Attack"

Seckin Anil Unlu
Department of Computer Engineering
TOBB University of Economics and Technology
Ankara, Turkey
sunlu@etu.edu.tr

Kemal Bicakci
Department of Computer Engineering
TOBB University of Economics and Technology
Ankara, Turkey
bicakci@etu.edu.tr

***Abstract*—In recent years phishing attacks have become one of the most important problems of online security. Aza Raskin, the creative lead of Mozilla Firefox team, proposed a new type of phishing attack, "tabnabbing attack" as he names it. The attack is different from classical phishing attacks; while classical attacks rely on deception of users with a similar URL and/or content in appearance to the original site, this attack uses our memory weakness and false perception that browser tabs are immutable i.e., do not change while inactive. We develop a Firefox add-on to protect users against this attack. Our method is based on the fact that a phishing web site should change its layout radically to look like the original site. This add-on watches the open tabs and indicates whether one changes its layout, favicon and/or title to become like another site.**

Keywords-browser add-on; phishing; software; tabnabbing; web security

I. INTRODUCTION

Phishing attacks are becoming more and more common and dangerous [1]-[3]. We all do so many things online, like banking, mailing, working and socially interacting with friends, etc. So thus we increasingly rely on the cloud. This fact makes phishing attacks so attractive and an easy way of deceiving people in order to make illegal profit. Many people going online are not informed computer users or experts, so they do not know what URL, SSL, JavaScript or HTML is and why these are related to their online security. People only know that they should pay attention to the "lock symbol" when they enter a web site where security is important, such as their banks'. But many of them do not pay attention to the address bar to check the URL [4]. These make things easier for attackers. They make their phishing sites look exactly similar to the original, use latest web technologies to deceive user within the right occasions such as attacking only some specific bank's customers which they can detect with CSS history attacks [5] or timing attacks [6]. Some other techniques include using international domain name support of new browsers to fake users with exactly similar looking domain names [7], [8]. Significant portion of spam mails are used for phishing attacks [10]-[12].

While these attacks become successful on many users, there are some basic precautions which prevent the attacks when people learn and put them into practice. But an attack recently proposed called "tabnabbing attack" is different [13]. While classical phishing attacks only work if the user doesn't know much about online security and acts carelessly, which we may call "the original deception"; this attack may succeed even if the user is tech savvy and well informed. It takes advantage of our perceived immutability of things we leave in place. A malicious site nabs its tab while the tab is not active and uses the tab to deceive the user to make her think the tab is for some other web site. So this nabbed tab can be used to get login information or for other phishing intentions. It works because we do not expect a page to change behind our backs and we do not pay attention to what is going on in unfocused tabs.

We develop a Firefox add-on which watches open tabs and warns the user when a malicious site nabs its tab and changes its content into a phishing page. We construct our add-on over the fact that the malicious page must change its layout and can be caught if we compare the two layout information.

II. THE TABNABBING ATTACK AND POSSIBLE IMPROVEMENTS OVER IT

A. Attack Details

A possible tabnabbing attack, as described by Aza Raskin, works as following:

- User navigates to a proper site which includes the attack script, but initially looks normal.

- The JavaScript embedded in the page detects when the page is not interacted with or waits for some predetermined duration.

- Then the script replaces the favicon and page title with the icon and title of the page attacker tries to imitate i.e. Gmail login page and transforms the page into something look-a-like.

- The user returns to the page when she looks over her open tabs and clicks to that tab. As she thinks she has left her page (Gmail application) open.

- She sees a login page and on the page there is a message like "Your session is expired" or "You are required to login again". She thinks she has been logged out and tries to login again providing her login information.

- The page captures login information and optionally redirects the user to the original mail page which the user is already logged in.

978-1-4244-7761-6/10/$26.00 ©2010 IEEE

The attack relies on human memory weakness and our perceived immutability of things we recently interact with. We could not exactly be sure about whether we have visited a web site or have been logged in. We perceive things we recently interact immutable and do not need to check the validity of them. Although there may be some minor changes or clues that may give away the attack we may not be able to detect these [14]. These facts make even the most tech savvy or paranoiac users susceptible to this attack.

The original attack requires scripting support which is already present and enabled by default on every modern web browser. JavaScript can be used to observe mouse and keyboard events to detect user activity on a page and to change the favicon [15], title and content of the page. Alternatively, researcher Aviv Raff made a proof-of-concept demo of this attack which does not require scripting support and uses HTML refresh *meta tag* [16]. So the page refreshes itself in the background and then suddenly turns into a phishing page.

Raskin also tells about how an attacker can improve the attack using several other techniques known. He says that one of them is CSS history attack [5] for which he has developed a tiny JavaScript library. This library can be used to detect which sites the victim visits, which mail provider or bank she uses, etc. Another effective way to detect these is the timing attack [7]. In Firefox 3.5 and beta versions after that, CSS attacks are no longer possible [17], but timing attacks are still applicable. He also reminds that the attacker could use *window.onerror* events to detect whether user is logged into a web site.

There are many ways to initiate the attack, such as third-party scripts, mashups, plugins -especially RIA plugins like Flash- and some known cross-site scripting (XSS) attacks [18]. These make the attack even more dangerous.

There are some really effective ways discovered to fool users even if they look at the address bar carefully. This attack, called homograph attack [7]-[9], uses international domain name (IDN) support on modern browsers, and by using such domain names in languages other than English but looking very similar to the original name, it's nearly impossible to tell the difference between the two addresses.

When these attacks are combined even the most paranoiac and skeptical user can be deceived.

B. Current Solutions

Aza Raskin's solution is the planned Firefox account manager [16]. With this manager, it will be possible for the browser to manage users' account information. The information will be available right near the address bar like the SSL certificate information bar in the current versions. This manager will get account information right from the login and signup pages, and it will show your session status on the bar. So users won't be likely to login to malicious phishing sites. The account manager will provide login information after once recorded and users hopefully will notice a malicious page when their login information doesn't appear. (It may be integrated with the password manager.)

There is a great Firefox add-on called NoScript [17] which blocks HTML objects, scripts and similar functionality on a domain basis. Unless you explicitly allow scripts for a specific domain, no permission is given them for execution and loading. This protects users from malicious scripts, tracking tools, XSS, CSRF (cross-site request forgery) and clickjacking attacks. This add-on protects you from these malicious scripts, and there is an update particularly about the HTML-only version of tabnabbing attack [18] which will keep tabs from refreshing themselves in the background. This prevents the attack conducted with scripting support but the attacker could still try the HTML-only way. Against tabnabbing attacks, there are some other shortcomings of this solution which are not directly related to the add-on's functionality. Not many users, especially non-power users are fond of it because it continuously asks the user whether to allow a page to execute scripts or not. Also it's not possible for the add-on to block a page after user has given permission to that page. The site could turn into a malicious one after getting the permission. If you don't configure the add-on in a really paranoiac way, you keep on trusting a page forever.

In theory, it's possible to get protection by using your browser's password manager [22]. This way you will notice if a login page is not automatically filled and hopefully look at the address bar. Many browsers provide a password manager or you can use one of many third-party password managers available freely (e.g., [28]).

C. Some Improvements on the Attack

In order to bypass the password manager solution, attacker could try to mimic the automatically filled form fields. This requires the username and/or password fields should be filled correctly (at least should appear to be filled correctly). So the attack works as following:

- A malicious page embeds the original version of the page in an *iframe* under the phishing page by giving the two *iframe*s full size and the same position.

- The page is designed in such a way that only original page's login form fields are seen from above. So the password manager fills these fields automatically (or after the correct entry of the master password) and then the user sees that the fields are filled with values (e.g. her account name). (Figure 1)

- Attacker has now two options:

1. He can try to fool the user by giving a message telling her password is wrong and then make her to type her password manually.

2. He can choose not to fill the password field right from the start, so the user may think that there is a problem with the password manager and she will type her password manually.

Both of the solutions can be implemented easily with positioning transparent and opaque form fields over the originals.

Some pages won't allow themselves to be displayed on iframes, but there is a quick solution to this, called frame buster buster [23]. Although there are solutions can be called frame buster buster busters [24], these are not used on many sites like Gmail.

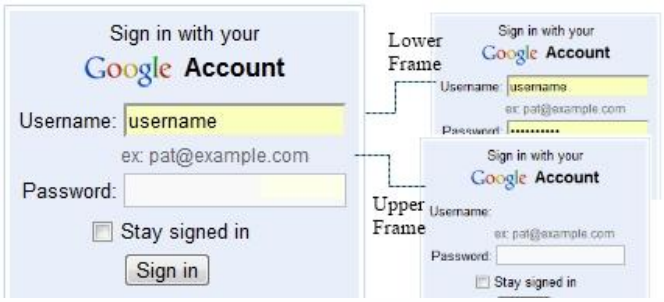

Figure 1. The attacker may prepare a page with two iframes where the malicious one above the original one. The original page makes the browser autofill the login form and the user doesn't notice any strangeness.

A new attack proposed by Jeremiah Grossman tries to steal form completion data from the browser [25]. This attack works on Internet Explorer 6/7 and Safari. It basically tries to mimic the user's keyboard dialing and sends some key stroke events to the browser via JavaScript. For example sending the key strokes DOWN, DOWN, ENTER usually activates the form completion list, selects one of the entries and closes the list. After that the attacker can read the auto filled data from the text box. In this way it's possible to reach users' data such as login name, e-mail, address, phone number, etc. This data may be used to autofill the login username or e-mail, and the user may be convinced that the form is autofilled by the autofiller and so it's genuine. Using this new attack, the attacker doesn't need to use any *iframe*s.

It's not required to have scripting enabled in order to change the favicon on Firefox, because it supports animated GIF favicon format. It's easy to design a favicon [26] which animates into another icon after waiting for some short period. One disadvantage of this method is that the duration is static so it has to be predetermined. We also note that this method is unavailable in other browsers which are not based on Gecko engine. It may be possible to carry out the attack with only using CSS when advanced CSS animation support will be available.

Before launching the attack an attacker may convince the user to enable scripting support using a compelling message in a page with some elements requiring scripting.

An interesting way to execute the attack may be the time when the user lefts her PC and walks out or she minimizes the browser and then engages into other applications. Attacker can determine when the current tab is not interacted with by using some JavaScript code which tracks keyboard and mouse events., When the attack is initiated when the focused page is not interacted with, probably the user is not able to notice the changed page although it is on focus. . This way, attacker can bypass NoScript protection which blocks page refreshes on only background tabs.

It's possible to launch the attack directly after the page is loaded by using some script to determine if the page has focus. So there may be no need to change the favicon and title. Most users open many background pages to read later while they are browsing on news sites or feed readers. Middle mouse button is generally used, so a script may open these links in direct phish mode when clicked on with middle mouse button. This type of attack is special to users who generally use their middle mouse button or some other shortcut to open a few number of pages in the background and browse them in order.

This attack mainly exploits the users' fallacious trust to things they seem familiar to them. Another similar application of this attack may be changing the function or mimicking the appearance of some well-recognized page elements using embedded or dynamically loaded JavaScript and do something malicious when the user consciously clicks on them. For example navigation and logout links, close buttons, login *iframe*s which appear in the page when required etc. may be used for this attack. Users will think that these links, buttons, frames are safe as always and will do only the function they were doing before, so they won't hesitate to click on. For this attack to work effectively some XSS vulnerabilities may be required. We can call this attack as "clicknabbing" or "functionnabbing". It's similar to the well-known clickjacking attack, but is not implemented in the same way. This type of attack may be used by advertising providers to deceive the user to receive extra clicks.

III. PROTECTION TECHNIQUE AND THE ADD-ON IMPLEMENTATION

Our protection technique is based on the fact that a phishing page which mimics another must change its title, favicon and layout. This may seem an extremely optimistic assumption, but in general, web pages are so complex and they include a rich set of elements such as images, links, input boxes, paragraphs, div and span elements which also have a rich style configuration. It's hard to find any two pages which have very similar layout except that these have the same style or template origin (such as these pages belong to the same domain or they use the same web application and theme). So if we track pages and calculate the level of change in the layout, we can detect if a page turns into another without refresh. The sensitivity to detect a radical layout change can be optimized to separate between more interactive pages and tabnabbers.

A. General Working of the Add-on

Our objective is to track favicon, page title and layout changes for every single tab until a new URI is loaded, and to inform the user if a radical change happens by comparing the old and new states when she activates one of these tabs.

When the user enters to a new web site or a new URI is loaded, favicon, page title and information about the layout of the page elements are recorded by the add-on. When she returns to that tab, new data got from the active tab is compared to the previously recorded data related to that tab. If there is a change in favicon and title or a change that's enough to trigger a warning in the layout, the user is informed by highlighting the address bar in yellow or red according to the warning level and by displaying a non-blocking warning message.

B. *Techical Details*

HTML page elements have offset left, top, width and height values which are calculated after embedded style information and CSS rules (Figure 2). These values can be changed with JavaScript and the attack has to do exactly this. There are also some important CSS style rules which determine whether an element is visible, on top of another, etc. As told above *iframe*s can be used to strengthen the attack, so either their position information or their children elements should be tracked. Some of the CSS style options that can be regarded as critical are positioning, float, display, padding, margin, border, opacity, visibility, background-image, background-position, repeat and size, transparent background-color and color, clear, overflow-x and y, z-index, etc.

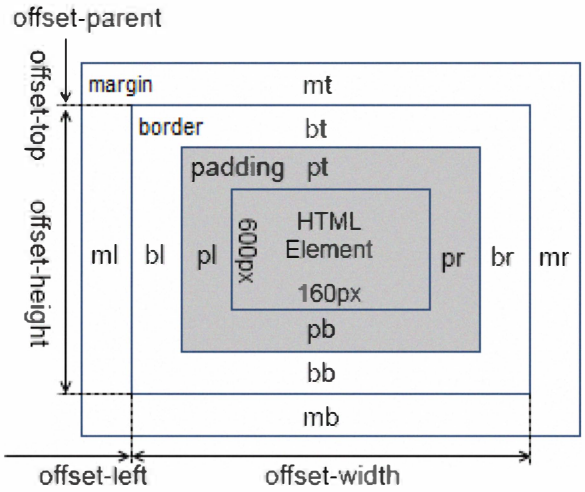

Figure 2. Every HTML element has some positioning properties as shown above. These can be used to calculate changes in the layout of the page. The figure shows exemplary margin, border, padding and offset values. Variables within the boxes define the amount of spacing for that specific layout property.

Our add-on does its work as following:

- Page titles and favicons are recorded for each tab. Values of the attributes stated above are recorded for each topmost page element on these tabs and critical CSS information is attached to these records.

- When focus gets from one tab into another, the operation is repeated and the add-on compares these values with the old state. If there is a difference between the title and favicon or a radical difference between the two layout information, it warns the user passively and highlights the address bar.

- If a page refreshes itself, redirects to somewhere else or loads a new URI, this recording process is repeated and compared to the old state.

The layout information is recorded for only the topmost elements and not for every page element, because it's easier to track and compare only the elements that effects the view users see. We get these topmost elements calling the method *document.elementFromPoint(x, y)* over some preselected grid points on the page (Figure 3).

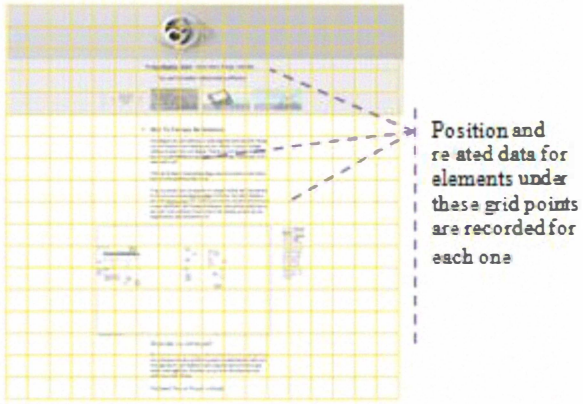

Figure 3. We use preselected grid points over the page body to compare the two states of the page layout for each browser tab.

This methodology is better than storing and comparing every element's data, because it has some clear advantages:

- The add-on must calculate elements' z-order (depth information) [27] and decide which one is above another to compare the two data correctly. But if we only compare the topmost elements, this problem is solved automatically.
- We have to match mutual elements between the original layout data and the new data by matching ids, tags or using coordinate information. But this process is harder than it seems and it's error prone. Some slight layout changes may cause wrong matching of mutual elements and this leads to more false positives. But if we only compare topmost elements over coordinate information, we can detect layout changes under grid points correctly. This approach makes it more resistant to minimal page layout changes which are frequent on dynamic web pages we use today.
- In this way we won't compare all elements on the page, so it is more resource-friendly. The attacker may also use this to confuse the add-on by adding extra dummy elements deliberately. If we compare only the topmost constant number of elements we avoid these problems.

There are also some problems of this technique. Because we don't keep data about all elements on the page, it may be possible to bypass the controls by putting very small invisible (1x1 px) elements under grid points and deceive the add-on. In order to overcome this problem, we should select grid points or the offset point randomly for each page. If the document contains many *iframe*s within each other, it's required to check the elements inside these recursively and compare them with the corresponding elements in the old data.

C. *Other Problems and Solutions*

Our technique is sufficient to detect layout changes, but there are some problems about page layout that's not related directly to the technique. For example if the user resizes her browser, some web pages are designed to re-layout themselves and some are not. This situation must be handled correctly.