Nabi - A Preventative Solution Against JavaScript and Scriptless Tabnabbing Attacks

Michael Baraboo, William Schattgen, Christopher To

4 May 2015

Abstract

1 Introduction

2 Background

2.1 TabShots

One approach to combating tabnabbing is a Google Chrome browser extension known as TabShots, which was created by Philippe De Ryck, Nick Nikiforakis, Lieven Desmet, and Wouter Joosen [1]. To detect whether a tab has changed or not while not in focus, TabShots uses visual comparison of the tab's appearance before and after the change in the user's focus. The extension records screenshots of the currently focused tab at regular intervals, using a simple Google Chrome API call and storing the screenshot as a data URL. The program also keeps record of the tab's favicon, or the icon displayed in the tab's url and title space. These two sets of images provide the basis for comparison when a tab regains focus.

When any tab regains focus, TabShots compares a new screenshot of the tab and it's current favicon to the most recent stored screenshot and favicon that was recorded before the tab lost focus. The small favicons are compared by source, and the screenshots are compared in a visually divided manner using the HTML5 canvas element. The screenshots are divided

in a raster of fixed-size tiles, and each tile in the new screenshot is compared to the most recently stored screenshot of the tab. The tiles are sized as 10x10 pixels, which are deemed by the authors as a balance between performance and precision. If the tiles do not match exactly, that area is marked as changed. The HTML5 canvas element provides powerful image manipulation capabilities that allowed the rastering and comparison algorithms to be implemented.

Once the differences in the current tab from its previous version are calculated, an overlay is injected into the page that shows the differences in semi-transparent red. The overlay is both transparent in both visibility and mouse events, so no mouse and keyboard events will be affected it. The overlay is constantly checked for its presence by the extension whenever an element is removed, ensuring that a malicious page will not be able to remove the overlay without notifying the extension and user. TabShots also places an icon onto the browser's toolbar that changes color in response the amount of changes on the currently focused tab. The icon serves as an unobtrusive security layer that will remain in view should the overlay be removed.

Finally, there is also an intended component for blacklisting sites that have confirmation of tabnabbing attempts. The developers have created an optaionl server-side component which can pull reported URLs from individual users, and add the confirmed URLs to a database. If the user decides that the current page is attempting tabnabbing and provides their explicit approval, the application can be signaled to send the accused page's URL, the image before the user switched tabs, and the image of the page after the user switched tabs. The server-side application uses its own automated comparison process and the further verification of a human analyst to decide if the page is indeed a phishing page, and whether the URL should be added to a subscription model blacklist.

According to the testings of the creators, TabShots is said to provide large compatibility with most major sites on the internet at a seemingly acceptable processing speed. The extensions was used on the top 1000 websites listed on Alexa.com, and the results showed

that 78% of these sites fall within the safe threshold of 5% or less in changed blocks, meaning no compatability issues exist with that page. About 19% of these sites have moderate changes less than 40%, which is the threshold for high amounts of changes, and 3% report more than 40% differences in changed blocks. While the last two numbers may seem high in terms of false positives, the authors note that TabShots never interferes with the functionality of a page, and does provide a whitelist to providing overlays for these sites. In terms of speed, testing showed that TabShots was able to prepare and calculate the image comparisions on a 1366×768 window within an average time of 284ms, which should provide little impact on browser operating time.

There are some problems with this application. First, the performance testing of TabShots was done on a 1366×768 window. Many users today have monitors with much larger resolutions such as 1920×1080 or 2560×1440, which provide two or four times as many pixels. This means that TabShots will run considerably longer on these types of monitors and provided a much larger performance strain. TabShots is designed to calculate the comparisons every time the user changes tab, which lead to very frequent occurrences of calculations depending on user behavior, and can further exaggerate TabShot's slower performance on higher resolution monitors. Finally, the visual comparison property of TabShots leads to many occurrences of false positives that lead to the high numbers mentioned above. Examples of non-malicious behavior that would trigger overlay occurrences would be videos, GIFs, image slideshows, overlays, and other dynamic advertisements. The authors note that slowly images and files would flag large amounts of tile changes. In addition, if an element was added or removed from the page, every other element on the page would shift, which would flag a major change in the picture comparison algorithm. The uncertain performance of TabShots on large monitors or frequently changing tabs, as well as its high opportunities for false positives make TabShots a potentially unideal solution to tabnabbing.

2.2 TabSol

Another attempt to detect tabnabbing is TabSol, a browser plugin created by Amandeep Singh and Somanath Tripathy [2]. TabSol uses a whitelist and hash computation to determine if a web page has changed while out of focus. TabSol attempts to detect tabnabbing using less computations and storage than other programs such as TabShots.

The process of TabShots begins with a simple whitelisting check, to check whether the newly opening URL has already been verified as legitimate. If so, the process execution is stopped. Otherwise, TabSol will calculate the hash digest of the web page source to save the current state of the page, using the cryptographic hash function SHA-1. When the user switches back to the old tab, TabSol will recompute the hash digest of the page, and compare it to the stored digest. If they match, TabSol determines that the site is legitimate and adds the page into the domain whitelist mentioned above. Else, the page is treated as suspicious, and TabSol attempts to find a login form on the changed page, as the authors consider its presence necessary in a phishing attack. The authors chose to sue the login form detction method presented in CANTINA+. If a login form is present, then the page is considered a phishing web page attempting tabnabbing, and appropriate actions can be taken.

The performance of this implementation is a strong feature of this browser extension. On an average machine with a 2.53 GHz processor, 4GB RAM and Windows 7 (64-bit), TabSol takes on average 157ms to detect a Tabnabbing attack, if it exists. The process only uses two hash computations, and the most time-consuming part is the login form detection, which only is used on a subset of tab changes where the hash computations do not match. In addition, the hash values are the only things needing to be stored, which takes up much less memory than screenshots of the user's windows in the case of TabShots. Finally, the authors make the case that the login form detection is a crucial component of TabSol that is lacking in other tabnabbing detection systems, and prevents the large amounts of false positives in TabSol tabnabbing detection.

The authors do leave some issues of TabSol unaddressed in their paper. First, the authors

make the fundamental assumption that any tabnabbing page will have a login form. This can be simply avoided by using a tabnabbing page that uses alternative means of communication such as a phone number, email address, or even a link to another login page. The login form detection system will completely fail in detecting these types of phishing attacks. Second, TabSol immediately and prematurely place any site that has matching hashes onto the whitelist, which could result in malicious sites being placed onto the whitelist. If the user switches away and back to the tab before the malicious tab is scheduled to change, the tab will not change at all, and it will pass TabSol testing and be placed on the blacklist. This login form basis and premature whitelisting can easily lead to the infiltration of tabnabbing sites.

2.3 NoTabNab

NoTabNab is a browser addon designed by Secking Anil Unlu and Kemal Bicakci. Their main objective was to focus on the favicon, page title, and layout changes for each tab, as the authors state that these properties must change in a tabnabbing attack. In particular, the program looks at important offset and margin values that are calculated for the page after the CSS style rules and properties are applied to each HTML element. The authors argue that any tabnabbing site must have some influence on these values, and thus they can be used to detect if the site has changed in an attempt of a tabnabbing attack.

The application begins by recording the page titles and favicons for each tab. NoTabNab then records the important layout values of the topmost page elements on these tabs. By topmost elements, the authors mean the elements which have the highest value on the z-axis above all other elements, as these are the elements that are most likely to change in a tabnabbing attack. When the tab focus returns to a tab, the stored values of the tab are compared to the new ones, and if any change is found, the user is passively alerted through highlighting the address bar. It should also be noted that if the page refreshes itself, redirects to somewhere else, or loads a new URL, the entire recording process is repeated

and compared to the older recorded state.

The authors point out the advantages of their minimal information gathering methodology. The primary benefits of only recording data about the topmost elements is the reduced work over gathering the necessary data points for every element, which results in less storage and processing to initially store the data, and less processing to compare the various data points for each element in the document. In addition, the comparison of values will be resource friendly, as only the simple favicon, title, and important CSS values are compared for each page, as opposed to entire high resolution pictures or entire documents.

The less thorough and structured methodology of NoTabNab does allow some problems in usability and tabnabbing tracking. The elements that are tracked by NoTabNab are chosen at random by finding the topmost elements of the page at random grid points. NoTabNab may be intentionally tricked into tracking nonchanging elements by placing very small, almost invisible elements that are only a handful of pixels large on the page. If these elements are scattered through the page at the highest z-order, then NoTabNab will track their changes, and not the changes of any other elements. In addition, if the document contains many iframes within each other, then the addon must recursively record and check the elements inside them, increasing the storage and processing requirements of the application. Finally, if the user resizes their window, any tab that resizes or changes their layout as a direct result of that action will appear to have radically changed under NoTabNab, and it will provide entirely false and ignorant data about the changes of the user's tabs.

3 Proposal and Methodology

4 Results

5 Conclusion

References

[1] Lieven Desmet Wouter Joosen Philippe De Ryck, Nick Nikiforakis. Tabshots: Client-side detection of tabnabbing attacks, 2013. 978-1-4503-1767-2.

[2] Amandeep Singh and Somanath Tripathy. Tabsol: An efficient framework to defend tabnabbing. IEEE, 2014. 978-1-4799-8084-0.