TabShots: Client-Side Detection of Tabnabbing Attacks

Philippe De Ryck, Nick Nikiforakis, Lieven Desmet, Wouter Joosen
iMinds-Distrinet, KU Leuven, 3001 Leuven, Belgium
Philippe.DeRyck@cs.kuleuven.be

ABSTRACT

As the web grows larger and larger and as the browser becomes the vehicle-of-choice for delivering many applications of daily use, the security and privacy of web users is under constant attack. Phishing is as prevalent as ever, with anti-phishing communities reporting thousands of new phishing campaigns each month. In 2010, tabnabbing, a variation of phishing, was introduced. In a tabnabbing attack, an innocuous-looking page, opened in a browser tab, disguises itself as the login page of a popular web application, when the user's focus is on a different tab. The attack exploits the trust of users for already opened pages and the user habit of long-lived browser tabs.

To combat this recent attack, we propose TabShots. TabShots is a browser extension that helps browsers and users to remember what each tab looked like, before the user changed tabs. Our system compares the appearance of each tab and highlights the parts that were changed, allowing the user to distinguish between legitimate changes and malicious masquerading. Using an experimental evaluation on the most popular sites of the Internet, we show that TabShots has no impact on 78% of these sites, and very little on another 19%. Thereby, TabShots effectively protects users against tabnabbing attacks without affecting their browsing habits and without breaking legitimate popular sites.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection; H.3.5 [**Information Storage and Retrieval**]: Web-based services

General Terms

Security

Keywords

Tabnabbing, phishing, client-side protection

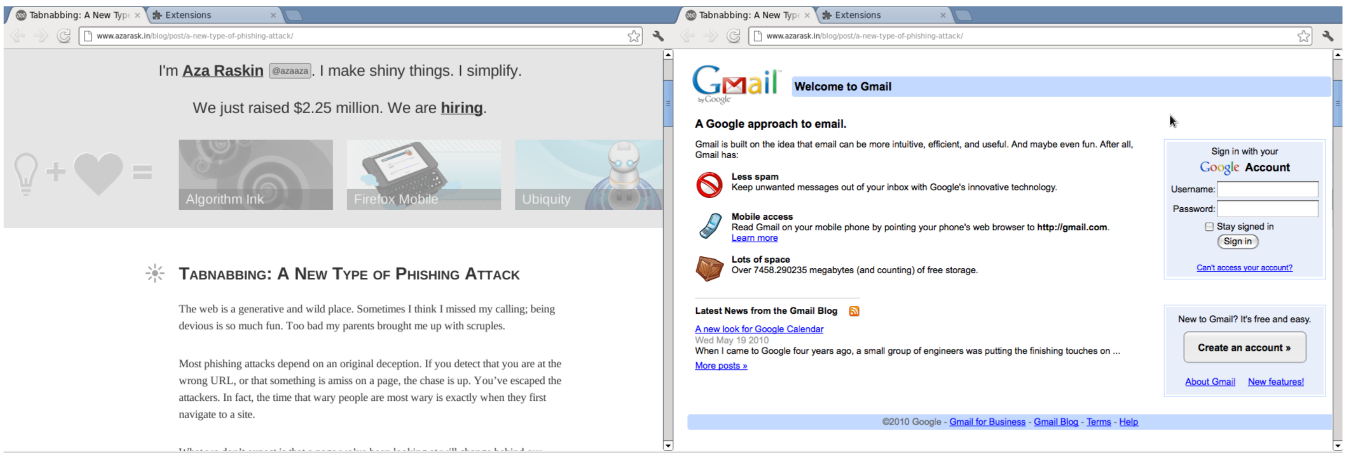
1. INTRODUCTION

Phishing, the process that involves an attacker tricking users into willingly surrendering their credentials, is as prevalent as ever. PhishTank, a volunteer-driven site for tracking phishing pages [13], in their latest publicly available report, reported a total of 22,851 valid phishing attempts just for July of 2012. In these attacks, an attacker targets the user and capitalizes on a user's inability of distinguishing a legitimate page from one that looks legitimate but is actually fraudulent. Phishing attacks can be conducted both on large and small scale, depending on an attacker's objectives. The latest publicized attack against the White House, involved the use of "spear phishing", a type of phishing that is targeting highly specific individuals and companies [9].

In 2010, Aza Raskin presented a new type of phishing attack which he called "tabnabbing" [14]. In tabnabbing, the user is lured into visiting a malicious site, which however looks innocuous. If a user keeps the attacker's site open and uses another tab of her browser to browse to a different website, the tabnabbing page takes advantage of the user's lack of focus (accessible through JavaScript as `window.onBlur`) to change its appearance (page title, favicon and page content) to look identical to the login screen of a popular site. According to Raskin, when a user returns back to the open tab, she has no reason to re-inspect the URL of the site rendered in it, since she already did that in the past. This type of phishing separates the visit of a site from the actual phishing attack and could, in theory, even trick users who would not fall victim to traditional phishing attacks.

In this paper we present TabShots, a countermeasure for detecting changes to a site when its tab is out of focus. TabShots allows a browser to "remember" what the tab looked like before it lost focus, and compare it with the appearance after regaining focus. More precisely, whenever a tab is fully loaded, TabShots records the favicon[1] and captures a screenshot of the visible tab. Whenever a user revisits a tab, a new capture is taken and compared to the previously stored one. If any changes are detected, the user is warned by adding a visual overlay on the current tab, showing exactly the content that was changed, assisting the user in distinguishing between legitimate changes and tabnabbing attacks. Our system is based on the user's visual perception of a site and not the HTML representation of it, allowing TabShots to withstand attacks that straightforwardly circumvent previously proposed, tabnabbing-detection systems. We implement TabShots as a Chrome extension and evaluate it against the top 1000 Alexa sites, showing that

[1]The small icon displayed in the tab's title space



I'm Aza Raskin @azaaza. I make shiny things. I simplify.

We just raised $2.25 million. We are hiring.

TABNABBING: A NEW TYPE OF PHISHING ATTACK

The web is a generative and wild place. Sometimes I think I missed my calling; being devious is so much fun. Too bad my parents brought me up with scruples.

Most phishing attacks depend on an original deception. If you detect that you are at the wrong URL, or that something is amiss on a page, the chase is up. You've escaped the attackers. In fact, the time that wary people are most wary is exactly when they first navigate to a site.

Welcome to Gmail

A Google approach to email.

Gmail is built on the idea that email can be more intuitive, efficient, and useful. And maybe even fun. After all, Gmail has:

Less spam
Keep unwanted messages out of your inbox with Google's innovative technology.

Mobile access
Read Gmail on your mobile phone by pointing your phone's web browser to **http://gmail.com**.
Learn more

Lots of space
Over 7458.290235 megabytes (and counting) of free storage.

Latest News from the Gmail Blog

A new look for Google Calendar
Wed May 19 2010
When I came to Google four years ago, a small group of engineers was putting the finishing touches on ...
More posts »

Sign in with your Google Account

Can't access your account?

New to Gmail? It's free and easy.

©2010 Google - Gmail for Business - Gmail Blog - Terms - Help

Figure 1: A seemingly innocuous page on the left performs a tabhabbing attack once the user switches focus, resulting in the page on the right [14].

78% of sites fall within a safe threshold of less than 5% changes, and an additional 19% fall within the threshold of less than 40% of changes. This means that TabShots effectively protects against tabnabbing attacks, without hindering a user's day-to-day browsing habits.

The rest of this paper is structured as follows: In Section 2 we first explore the original tabnabbing attack and then discuss possible variations taking advantages of the different implementations of the tabbing mechanism in popular browsers. In Section 3 we describe in detail the workings of TabShots and our implementation choices. In Section 4 we evaluate TabShots on security, performance and compatibility against the Alexa top 1000. In Section 5, we briefly describe how TabShots could be deployed on the server-side to create tabnabbing blacklists and expand protection to all users. In Section 6 we discuss the related work and conclude in Section 7.

2. BACKGROUND

2.1 Anatomy of a tabnabbing attack

Tabnabbing relies on the tab mechanism, which is common in all modern browsers. Users visit websites, but instead of navigating away from that website when they want to consume the content of a different website, they open a new tab, and use that tab instead. The old site remains open in the old tab, and many tabs can accumulate over time in a user's browser. A 2009 study of user's browsing habits revealed that users have an average of 3.2 tabs open in their browsers [6]. We expect that today, this number has increased, due to the sustained popularity of social networking sites and web applications that constantly update a user's page with new information. The latest features introduced by browsers attest to this popularity of multiple open tabs, since they give the user the ability to "pin" any given tab to the browser and treat it as a web application.

The steps of a tabnabbing attack as presented by Raskin [14] are the following:

1. An attacker convinces the user to visit a website under his control. This website appears to be an innocuous site that is not trying to fool the user into giving up her credentials. What the attacker must do, is convince the user to keep this tab open, and browse to a different website. This is easily achieved in a wide range of ways, for instance by providing an article that is both very interesting, but also too long to read in a single go, or some sort of free product that will be available in the near future. Directing the user away from the attacker's site is straightforward by adding the `target="_blank"` attribute to interesting hyperlinks, so that new links automatically open in a new tab or window.

2. JavaScript code running in the attacker's website is triggered when the current window has lost focus, by registering to the `window.onBlur` event handler.

3. The user keeps the attacker's website open and uses other tabs to surf the Internet.

4. The attacker realizes that his window is currently not in focus, and, after a possible delay of a few seconds in order to make sure that the user is busy consuming other content, changes the title, favicon and layout of the page to mimic the login screen of a web application, for instance the user's web mail or social networking site. The attacker can choose a default web application (like Gmail) under the assumption that most users have a Gmail account or can combine the tabnabbing attack with a history-revealing attack [8, 21], and present the login of a web application that he knows is visited in the past by the user. This process is also shown in Figure 1

5. At some point in the future, the user recognizes a tab with a familiar favicon (e.g. GMail) and unwittingly opens the attacker-controlled tab. At this point, the user is no longer checking the URL of the website, since it is a website that she opened in the past and thus "trusted". Given a convincing login screen, the user proceeds into typing her credentials in the given forms which are then transferred to the attacker, thus completing the tabnabbing attack.

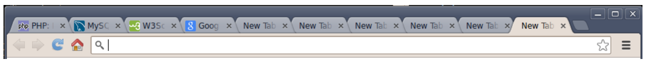


Figure 2: Chrome keeps all tabs visible but shrinks the space alloted to each tab

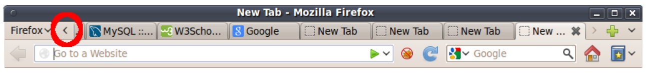


Figure 3: After a number of tabs, Firefox hides older tabs in order to make space for the new ones

The main difference between tabnabbing and traditional phishing attacks is that the fake login form is decoupled from the visit of the malicious website. Thus, users who have been trained to spot phishing attacks by immediately checking the URL of the page they open, may fall victim to this variant of phishing. This "delayed maliciousness" can also be used to evade detection by any automated honeyclients which may be autonomously searching for phishing pages based on various heuristics [22]. If the honeyclient does not stay for long enough on the malicious page, or does not trigger the `window.onBlur` event, then the actual phishing page will never be shown and the attacker can avoid detection.

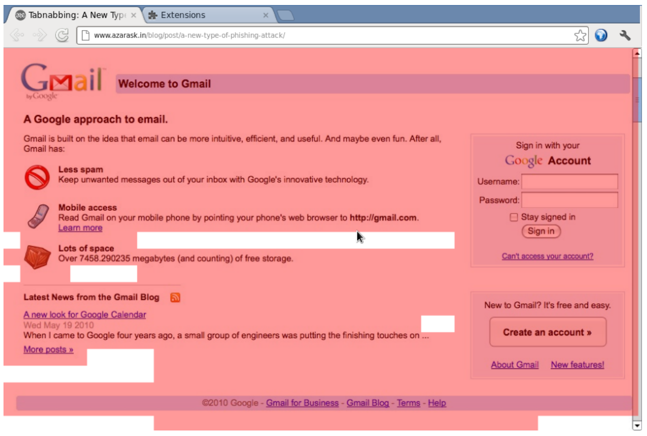
2.2 Overly Specific Detection

In the previous section, we described the anatomy of a tabnabbing attack, exactly as it was first presented by Raskin in 2010 [14]. According to Raskin, an attacker needs to change three things in order to conduct a successful tabnabbing attack: the page's title, the page's favicon and the page itself. Accordingly, currently known countermeasures depend on changes in these three properties, or include even more specific tabnabbing characteristics (more details in Section 6). This overly specific detection gives the attacker more flexibility to avoid detection.

One example of such flexibility is carrying out a tabnabbing attack without changing the title of the tab, simply by taking advantage of the tabbing behavior within a browser. While conducting our research, we noticed that different browsers behave differently when a user has many open tabs in one window. Figures 2 and 3 show how Chrome and Firefox handle many open tabs. Chrome, starts resizing the label of each tab, in an effort to keep all tabs visible. Here, one can notice that most of the title of each tab is hidden while favicons remain visible. On the other hand, Firefox starts hiding tabs which the user can access by clicking on the left arrow (circled in Figure 3). Moreover, Firefox preserves the title bar above the tabs, which Chrome dispenses in an effort to maximize the amount of space available for HTML.

In the case of Chrome, assuming that a user has many tabs open, the attacker can avoid the title change altogether, since it will likely not be visible to the user anyway.

In the next section, we present TabShots, which detects tabnabbing attacks using visual comparison. Since TabShots does not depend on fine-grained detection properties, we leave no room for an attacker to sneak through.



Welcome to Gmail

A Google approach to email.

Gmail is built on the idea that email can be more intuitive, efficient, and useful. And maybe even fun. After all, Gmail has:

Less spam
Keep unwanted messages out of your inbox with Google's innovative technology.

Mobile access
Read Gmail on your mobile phone by pointing your phone's web browser to **http://gmail.com**.
Learn more

Lots of space
Over 7458.290235 megabytes (and counting) of free storage.

Latest News from the Gmail Blog
A new look for Google Calendar
Wed May 19 2010
When I came to Google four years ago, a small group of engineers was putting the finishing touches on ...
More posts »

Sign in with your Google Account
Username:
Password:
☐ Stay signed in
Sign in
Can't access your account?

New to Gmail? It's free and easy.
Create an account »
About Gmail New features!

©2010 Google - Gmail for Business - Gmail Blog - Terms - Help

Figure 4: The overlay generated by TabShots for the attack from Figure 1. Here, only certain parts of the white background remained unchanged.

3. TabShots PROTOTYPE

3.1 Core idea

As discussed before, a successful tabnabbing attack depends on the user visiting a malicious page, shifting focus to a different tab and returning at some point, after which the malicious page has changed its looks to resemble a popular application's login form. In itself, a tabnabbing attack is extremely obvious to detect, since a convincing phishing page will differ from the previous content. Detection is however complicated by the tab being out of focus, and the user placing some trust in previously opened and visited tabs.

TabShots takes advantage of these obvious changes needed by a successful tabnabbing attack, by remembering what a tab looks like before it loses focus, and comparing that to what it looks like when it regains focus. Any changes that happened in the background will be detected, and communicated to the user by means of a colored overlay. This allows the user to decide for herself whether the changes are innocent (e.g. an incoming chat message) or malicious masquerading (e.g. a login form and GMail logo popping up). Figure 4 shows how TabShots detects the tabnabbing attack from Figure 1. This non-intrusive behavior guarantees compatibility with all existing sites, since changes are only highlighted and not blocked or prevented.

Our approach is purely built on the visible content of a tab, exactly as the user perceives it. This yields several advantages compared to techniques analyzing the structure and contents of a page. TabShots is invulnerable to HTML, CSS or JavaScript trickery, aimed at circumventing tabnabbing countermeasures (see Section 6), scrolling attacks or other obfuscation attacks.

3.2 Implementation details

TabShots is currently implemented as an extension for Google Chrome[2], but could easily be ported to other browsers supporting an extension system, provided they offer a reliable way to capture screenshots of tabs.

[2]A prototype of TabShots is available at `http://people.cs.kuleuven.be/~philippe.deryck/papers/asiaccs2013/`

In the following paragraphs, we discuss several implementation techniques and strategies for the major components of TabShots.

Capturing Tabs.

TabShots records the favicon and captures screenshots of the currently focused tab at regular intervals, keeping track of the latest version. This latest snapshot will be the basis for comparison when a tab regains focus. Capturing a screenshot of a tab in Google Chrome is trivial, since the browser offers an API call to capture the currently visible tab of a window. Capture data is stored as a data URL [10].

Capturing snapshots of a tab at regular intervals is a deliberate design decision, allowing TabShots to handle changes that happen in a tab while it is in focus. These changes typically occur in highly dynamic applications, such as Facebook or GMail, which often use AJAX techniques to dynamically update the contents of their pages. Ideally, a tab could be captured right before it loses focus, but since Google Chrome does not offer such an event, this feature cannot be implemented without a severe usability and performance penalty.

Comparing Tab Snapshots.

When a tab regains focus, TabShots needs to compare the current snapshot data with the stored data and detect any differences. Favicons are compared by source, and the screenshots are compared visually. Each screenshot is divided in a raster of fixed-size tiles (e.g., 10x10 pixels). Each tile is compared to its counterpart in the stored snapshot data. If the tiles do not match exactly, the area covered by it is marked as changed. The rastering and comparison algorithms are implemented using the recently introduced HTML5 canvas element, which offers extensive image manipulation capabilities.

One potential disadvantage of the screenshot analysis is the difficulty to detect a small change in a page that results in a visible shifting of contents (e.g. adding one message in front of a list). Such *false positives* may be addressed by a smarter comparison algorithm, that is able to detect movements within a screenshot.

The evaluation section (Section 4) discusses the chosen tile size and performance of the comparison algorithm in more detail.

Highlighting Differences.

Once the differences for a focused tab are calculated, TabShots injects an overlay into the page. This overlay is completely transparent, except for the differences, which are shown in semi-transparent red. The overlay is positioned in the top left corner and covers the entire visible part of the site. Setting the CSS directive *pointer-events: none* ensures that the overlay does not cause any unwanted interactions, and allows mouse and keyboard events to "fall through" the overlay onto the original content.

In order to detect a malicious page from actively trying to remove the overlay from the DOM, we implement a mutation event listener that is triggered when an element is removed. It then checks whether the overlay is still present and if not, immediately warns the user of this active malicious behavior.

Security Indicator.

In addition to the overlay of the changes on the current page, TabShots also has a browser toolbar icon, indicating the current status of the site. The icon's background color indicates how much of the site has changed, ranging from almost nothing ($< 10\%$, shown as green), over moderate ($< 40\%$, shown as yellow) to high ($> 40\%$, shown as red). Clicking on the icon shows a miniature view of the current tab combined with the overlay of detected changes. Having a security indicator as part of the browser environment ensures that even if a malicious page somehow manipulates or removes the overlay, the user still has a trustworthy notification mechanism.

The current notification mechanism is quite subtle, but follows other commonly accepted and implemented notification mechanisms, such as displaying a padlock when using a secure connection. If desired, the notification mechanism can be easily extended to something more visible, such as the warnings given in case of an invalid SSL certificate.

3.3 Alternative Design Decisions

During the design and development of TabShots, we considered different paths and options, leading to the outcome described here. For completion, we want to discuss two topics that drove the design and workings of TabShots in a bit more detail.

JavaScript-based Detection.

Instead of visually comparing screenshots, we might attempt to detect the malicious JavaScript code actually carrying out the tabnabbing attack. This is not a trivial task, since JavaScript's dynamic nature makes script analysis difficult. Furthermore, there are a multitude of ways to actually implement a tabnabbing attack. The attack example discussed earlier uses the `window.onBlur` event, but a tabnabbing attack is certainly not limited to only this event. Similarly, there are numerous ways to actually change the displayed content, ranging from the use of JavaScript to extensive use of available CSS techniques.

Regularly Capturing Tabs.

Currently, TabShots makes a capture of a tab at regular intervals, so it can compare the capture taken when the user returns to a fairly recent capture from before. Ideally, we would make a capture when the user leaves, and a capture when the user returns. Unfortunately, Chrome does not trigger an event when a user leaves a tab, only when a user focuses a new tab. At the moment this event is received, the new tab is already displayed. To take a screenshot of the tab that was just left, TabShots has to switch it back into display, take a capture and switch back to the new tab. Unfortunately, this cannot be implemented without very briefly revealing this process visually to the user, with a degraded user experience as a consequence.

4. EVALUATION

As discussed before, a tabnabbing attack takes place when a user leaves a innocuous-looking malicious tab unfocused. Tabnabbing is different from traditional phishing, since it exploits trust placed in a previously opened tab, whereas phishing simply tries to mislead the user.

Our evaluation of TabShots consists of three parts. First we discuss how TabShots effectively protects against all tabnabbing attacks. Second, we discuss the performance impact

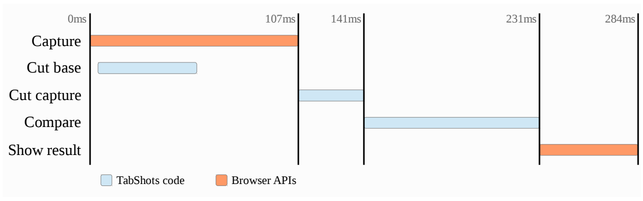


Figure 5: Breakdown of the average performance with a resolution of 1366x768.

of TabShots. The third part elaborates on the setup and results of an experimental compatibility study using Alexa's 1000 most popular sites.

4.1 Security

The security guarantees offered by TabShots follow directly from its design. We recapitulate the three most important security properties here: (i) zero false negatives, (ii) user-friendly and clear overlay and (iii) secure toolbar indicator.

TabShots can not miss a tabnabbing attack by design, since it visually captures screenshots from a tab and compares them. In order for a tabnabbing attack to occur undetected, it has to ensure that the screenshots before and after losing focus are identical, meaning the page did not change while out of focus. This case is considered a classic phishing attack, and not a specific tabnabbing attack.

Second, TabShots injects an overlay of the focused tab, indicating which parts of the page have changed since its last focus. Using mutation events, TabShots detects if a malicious page actively tries to remove the overlay, and notifies the user with a strong security message.

Third, TabShots also adds an icon to the browser toolbar. Using a three-level color indication system, it notifies the user of how much a tab did change. The strength of this toolbar icon is that it runs in the context of the extension, and is completely out of reach to any page-specific code. This effectively prevents any manipulation by a malicious page.

4.2 Performance

In order to prevent tabnabbing attacks, TabShots must be capable of warning the user of any changes *before* she enters any sensitive information. Furthermore, since TabShots's algorithm is executed when a user switches tabs, it is crucial that there is no noticeable performance impact. The performance measurements and analysis of the main algorithm, discussed below, show that TabShots succeeds in quickly processing the captures and warning the user of any changes that occurred.

One important advantage of TabShots is that it fully operates in the background, without any blocking impact on any browser action or processing. When a user switches tabs, TabShots will perform the following steps:

1. Capture a screenshot of the newly focused tab
2. Cut the previously captured image of this tab (before it lost focus) into tiles
3. Cut the newly acquired screenshot into tiles
4. Compare the tiles of both screenshots and mark the differences
5. Inject the calculated overlay into the page and update the TabShots icon

For a browsing window with a resolution of 1366x768, the most common resolution at the time of this writing [18], TabShots is capable of performing these steps within an average time of 284ms after receiving the browser event fired by switching tabs. Fig. 5 shows a breakdown of this time into the steps mentioned before. Note that of these 284ms, 160ms are consumed by browser APIs, which are out of our control.

Currently, a large chunk of time is consumed by the comparison algorithm, which is a pixel-by-pixel comparison of each tile. The time used by this algorithm is strongly correlated to the number of changes within a page. If a difference between tiles is detected at the first pixel, there is no need to check the remaining pixels. Consequently, if a tabnabbing attack occurs, a lot of changes will be detected and TabShots's algorithm will perform even faster. Table 1 presents the number of milliseconds spent on comparison on our testing pages, where we use a div to change a certain percentage of a page, clearly showing the correlation between amount of changes and required processing time.

% changes	ms spent on comparison
0	126
25	86
50	60
75	32
100	4

Table 1: Correlation between amount of changes on a page and number of milliseconds consumed by the comparison algorithm.

Overall, one can see that TabShots is efficient enough to prevent tabnabbing attacks, before the user discloses her credentials to the phishing page and without a negative effect on the user's browsing experience. Moreover, if TabShots was to be implemented directly within the browser instead of through the browser's extension APIs, we expect that its overhead would be significantly lower.

4.3 Compatibility

Apart from the security guarantees offered by TabShots, its compatibility with existing sites is another important evaluation criterion. When using non-malicious web applications, the number of changes detected by TabShots, i.e. false positives, should be limited, even though the user can quickly determine whether a change is legitimate or not.

To determine the compatibility with current web applications, we ran TabShots on the top 1000 Alexa sites. Each site was loaded in a separate tab, and captured before and after it lost focus. These two captures were compared and analyzed for the number of changed blocks. Through our preliminary experimentation with TabShots, we discovered that a 10x10 tile-size strikes the desired balance between performance and precision. Smaller tiles would incur extra overhead, since as the number of tiles increase, so do the checks between the old versions and the new ones, without

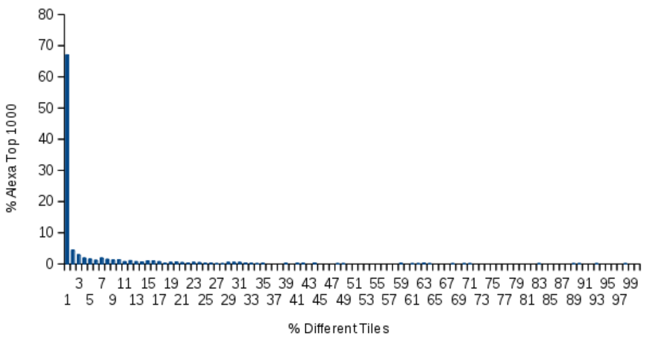


Figure 6: Compatibility analysis of the visual comparison algorithm with Alexa's top 1000 sites.

Domain	% of changed tiles
facebook.com	0.38
google.com	0.00
youtube.com	4.05
yahoo.com	5.31
baidu.com	0.00
wikipedia.org	0.73
live.com	2.65
twitter.com	2.91
qq.com	6.00
amazon.com	2.57
blogspot.com	0.32
linkedin.com	0.26
taobao.com	0.49
google.co.in	0.00
yahoo.co.jp	4.13
sina.com.cn	1.24
msn.com	23.22
google.com.hk	0.00
google.de	0.00
bing.com	0.00

Table 2: Compatibility analysis of the Alexa top 20 sites

a distinguishable improvement in pin-pointing the modified content.

Table 2 shows the results for the top 20 sites, and Fig. 6 shows a histogram of the entire top 1000, grouped by integer percentage values. The results show that 78% of sites fall within the safe threshold of less than 5% changed blocks, meaning there are no compatibility issues here. About 19% of sites have moderate changes, but still less than 40%. Manual verification shows that these changes are mainly caused by changing content such as image slideshows or dynamic advertisements. A typical example of an overlay of a dynamic advertisement is shown in Fig. 9. Finally, 3% of sites has more than 40% of changed blocks, which seem to be caused by changing background graphics. Fig. 7 and 8 respectively show the worst case scenario for the sites with moderate changes (less than 40%) and sites with heavy changes (more than 40%).

Note that even though certain sites have a high number of changed blocks, TabShots never interferes with a page, preventing any loss of functionality. If desired, a user can

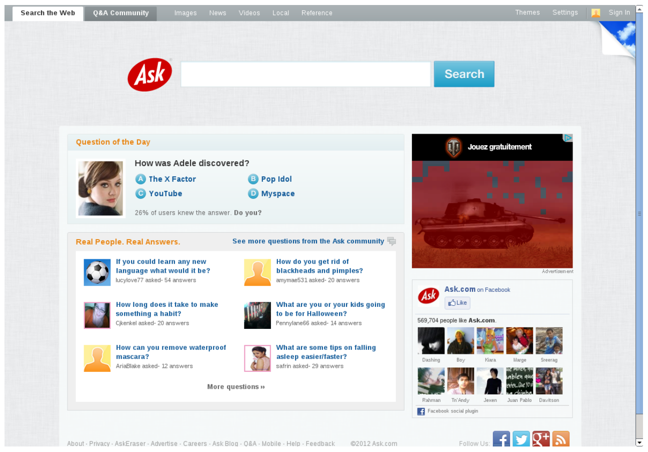


Figure 9: Screenshot of a typical dynamic advertisement being recognized by TabShots.

easily whitelist known trusted sites, to prevent needless overlaying of changed content. Additionally, a future extension of TabShots can incorporate a learning algorithm to identify dynamic parts of a site while the tab is in focus, which reduces the number of false positives.

The automated analysis gives a good idea of the impact on Alexa's top 1000, but is unfortunately not able to cover the authenticated parts of the sites. Therefore, we also tested the impact of TabShots on the daily use of several highly dynamic web applications, for example social networking applications (e.g. Facebook, Twitter) and webmail clients (e.g. GMail, Outlook Web Access). One noticeable effect is that the addition of a single element can cause a shifting of content within a page, which is currently flagged as a major change by the comparison algorithm. In future work, we can implement a comparison algorithm that detects such shifts and only marks the newly added content as a change.

5. BLACKLISTING TABNABBING

In the previous section, we described in detail the idea and implementation of TabShots. While this is sufficient for the protection of a user who has installed our browser extension, we deem it desirable to also protect users who are using different browsers or have not installed TabShots. This can be achieved through an optional server-side component which can aggregate information sent by individual browsers and, after validation, add the reported URLs in a blacklist. This server-side component would be the logical next step, to transition from a protection of a selected number of users (those with TabShots installed) to a more global protection, similar to Google's SafeBrowsing list of malicious sites [15] which is currently utilized by many modern browsers. In the rest of this section, we describe the possible workings of such a service.

In the stand-alone version of TabShots, once a user realizes that she is being targetted by a tabnabbing attack, she is instructed to simply navigate away from the malicious site without entering any private information. With a server-side component in place, the user can mark the current page as a "tabnabbing attack" through the UI of our extension.

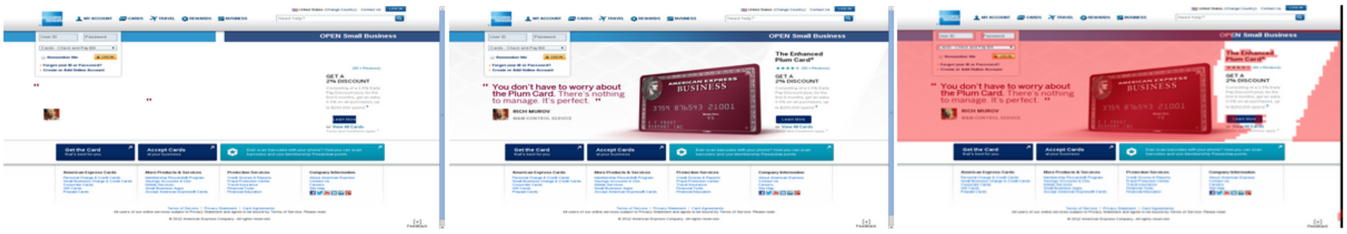


Figure 7: Before and after shots of *americanexpress.com (#354)*, which has 38.93% of changed blocks, due to a background image that took longer to load.

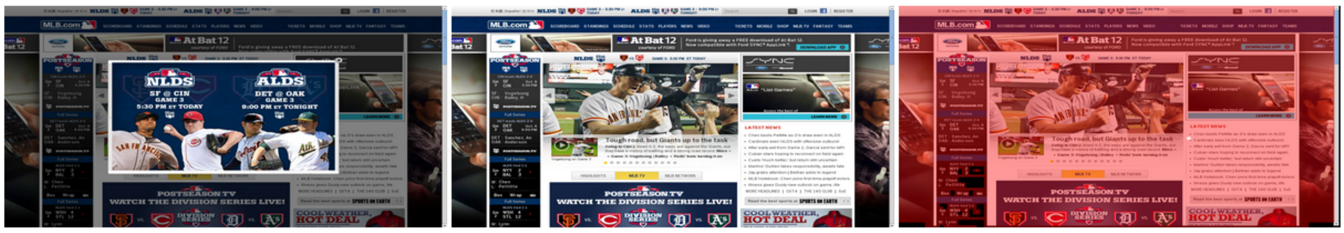


Figure 8: Before and after shots of *mlb.com (#355)*, which has 97.31% of changed blocks, due to an overlay that changed the intensity of the site to present an advertisement.

Once this happens, TabShots transfers to a server-side, data aggregator the following information:

1. The URL of the current page
2. The image of the page before the user switched tabs
3. The image of the page after the user switched tabs

The server-side aggregator has the responsibility of receiving reports from multiple users, filtering-out false reports and then adding the true positives on a blacklist. Filtering is necessary to stop attackers who wish to harm the reputation of the TabShots blacklist, by submitting legitimate sites that would then be automatically blocked. Our server-side service operates as follows:

For every previously unreported URL received by a user with TabShots installed, our service spawns an instrumented browser which visits the reported URL and captures a screenshot. Assuming that the current page is indeed performing tabnabbing, the malicious scripts will try to get information about their "visibility" through the `window.onBlur` event. Since our browsers are instrumented, we can trigger a `window.onBlur` event without requiring the actual presence of extra tabs. In the same way, any callbacks that the script registers using the `setTimeout` method, are immediately triggered, i.e., the malicious code is tricked into performing the tabnabbing attack, without the need of waiting. Once the callbacks are executed, our system takes another snapshot of the resulting page. The set of screenshots captured by the user is then compared with the set that was captured by our system. To account for changes in the pages due to advertisements and other legitimately-dynamic content, the screenshots are accepted if either the server-generated set is an identical copy of the user-generated one, or if they match over a certain configurable threshold (i.e. everything matches except certain dynamic areas).

Once the above process is complete, the URLs recognized as true positives are then sent to a human analyst who will verify that the resulting page is indeed a phishing page. A human analyst is necessary since our system cannot reason towards the maliciousness or legitimacy of the final changed page. Note, that human-assisted phishing verification is currently one of the most successful approaches, e.g., PhishTank [13], and its results are more trustworthy than any automated phishing-detection solution. The URLs that are marked as "tabnabbing", can then be added to a blacklist that browsers can subscribe to.

The users who report URLs that either never reach a human analyst (because the server-side screenshots did not match the user-provided ones), or reached a human analyst and were classified as "non-phishing" are logged, so that if they are found to consistently submit false positives, our system may adapt to ignore their future submissions.

Submitting screenshots to a third party service might be considered privacy-sensitive, so we take care to address these issues accordingly. Therefore, TabShots only submits a screenshot after explicit user approval. Additionally, the screenshot submission is only triggered after the user flagged a tabnabbing attack, so it is very likely that the captured site is malicious of nature, and does not contain any sensitive information.

6. RELATED WORK

Raskin was the first to present the tabnabbing attack in 2010 [14]. Others, presented variations of the attack, for instance redirecting the user through a meta-refresh tag to a new page, instead of changing the existing page with JavaScript [1], which would circumvent protections such as NoScript [12]. This attack however does not depend on user activity, but rather on a predefined timeout, by which the attacker hopes that the victim will have changed tabs and thus will not notice the changing web-site.

Unlu and Bicakci [20] proposed NoTabNab, a browser extension that monitors tabs in search for tabnabbing attacks. When a page loads, the extension records the title of the page, the URL, the favicon and several attributes of the topmost elements[3], as determined by the browser API call `document.elementFromPoint(x,y)`. While this approach is conceptually similar to ours, NoTabNab suffers from several issues that render it ineffective.

A first issue is that by capturing a tab when it is loaded, the extension will miss all content that is added dynamically (e.g. using AJAX) between loading a tab and actually switching to another tab. Second, the design of the detection mechanism offers an attacker several ways to evade it. For instance, an attacker can place all of the page's content in an iframe that spans the entire visible window. The `document.elementFromPoint` cannot "pierce" through the iframe, and will always return the iframe element, regardless of any changes that may happen inside the iframe. Another possible bypass, is through the overlay of a transparent element, that again stretches the entire page's content and allows clicks and interactions to "fall-through" to the actual phishing form under it. On the contrary, TabShots uses the screen-capturing API of the browser and is essentially using the same data as actually seen by the user. This design decision makes TabShots invulnerable to the aforementioned bypasses.

Suri et al. [19] propose the detection of tabnabbing through the use of tabnabbing "signatures". The authors claim that the combination of certain JavaScript APIs with HTML elements are tell-tale signs of a tabnabbing attack, and present two signatures, based on the presence of `onBlur`, `onFocus`, and other events within an iframe. Unfortunately, the authors make no attempt to characterize the false positives that their system would incur. Additionally, the presence of an iframe is by no means necessary for a tabnabbing attack. JavaScript code is capable to drastically change the appearance of a page through the addition and removal of styled HTML elements, thus allowing an attacker to bypass the authors' monitor. TabShots on the other hand, does not depend on anything other than the visual differences between the old and the new version of the tab, and thus will detect all visible changes, regardless of the technical means through which they are achieved.

While tabnabbing is a relatively new phishing technique, attackers have been trying to convince users to voluntarily give up their credentials for at least the last 17 years [3]. Several studies have been conducted, trying to identify why users fall victim to phishing attacks [5, 7] and various solutions have been suggested, such as the use of per-site "page-skinning" [4], security toolbars [23], images [2, 17], trusted password windows [16], use of past-activity knowledge [11] and automatic analysis of the content within a page [24]. Unfortunately, the problem is hard to address in a completely automatic way, and thus, the current deployed anti-phishing mechanisms in popular browsers are all black-list based [15]. The blacklists themselves are either generated automatically by automated crawlers, searching for phishing pages on the web [22] or are crowdsourced [13].

[3]When seeing a page as a stack of elements, the topmost elements potentially overlay other elements

7. CONCLUSION

Tabnabbing attacks are a type of phishing attacks where the attacker exploits the trust a user places in previously opened browser tabs, by making the malicious tab look like a legitimate login form of a known web application. This happens when the user is looking at another tab in the browser, making it very hard to detect and very easy to fall victim to.

Currently available countermeasures typically depend on several specific characteristics of a tabnabbing attack, and are easily bypassed or circumvented. Our countermeasure, TabShots, is the first to do a fully visual comparison, detecting any changes in an out-of-focus page and highlighting them, aiding the user in the decision whether to trust this page or not.

Our evaluation shows that TabShots protects users against potential tabnabbing attacks, with a minimal performance impact. Furthermore, an experimental evaluation using Alexa's top 1000 sites shows that 78% of these sites fall within the safe threshold of less than 5% changes in subsequent snapshots. This means that TabShots is fully compatible with these sites, and has very little impact on another 19%.

8. ACKNOWLEDGEMENTS

This research is partially funded by the Research Fund KU Leuven, IWT and the EU-funded FP7 projects NESSoS and Web- Sand.

With the financial support from the Prevention of and Fight against Crime Programme of the European Union European Commission – Directorate-General Home Affairs. This publication reflects the views only of the authors, and the European Commission cannot be held responsible for any use which may be made of the information contained therein.

9. REFERENCES

[1] E. Adler. Tabnabbing without JavaScript . `http://blog.eitanadler.com/2010/05/tabnabbing-without-javascript.html`.

[2] N. Agarwal, S. Renfro, and A. Bejar. Yahoo!'s Sign-in Seal and current anti-phishing solutions.

[3] AOL acts to thwart hackers. `http://simson.net/clips/1995/95.SJMN.AOL_Hackers.html`.

[4] R. Dhamija and J. D. Tygar. The battle against phishing: Dynamic security skins. In *Proceedings of the 2005 symposium on Usable privacy and security*, SOUPS '05, pages 77–88, New York, NY, USA, 2005. ACM.

[5] R. Dhamija, J. D. Tygar, and M. Hearst. Why phishing works. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, CHI '06, pages 581–590, New York, NY, USA, 2006. ACM.

[6] P. Dubroy. How many tabs do people use? (Now with real data!). `http://dubroy.com/blog/how-many-tabs-do-people-use-now-with-real-data/`.

[7] S. Egelman, L. F. Cranor, and J. Hong. You've been warned: an empirical study of the effectiveness of web browser phishing warnings. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '08, pages 1065–1074, New York, NY, USA, 2008. ACM.

[8] D. Jang, R. Jhala, S. Lerner, and H. Shacham. An empirical study of privacy-violating information flows in

JavaScript Web applications. In *Proceedings of CCS 2010*, pages 270–83. ACM Press, Oct. 2010.

[9] J. Leyden. Hackers break onto White House military network. `http://www.theregister.co.uk/2012/10/01/white_house_hack/`.

[10] L. Masinter. The "data" url scheme. 1998.

[11] N. Nikiforakis, A. Makridakis, E. Athanasopoulos, and E. P. Markatos. Alice, What Did You Do Last Time? Fighting Phishing Using Past Activity Tests. In *Proceedings of the 3rd European Conference on Computer Network Defense (EC2ND)*, volume 30, pages 107–117, 2009.

[12] NoScript - JavaScript/Java/Flash blocker for a safer Firefox experience! `http://noscript.net/`.

[13] PhishTank | Join the fight against phishing. `http://www.phishtank.com`.

[14] A. Raskin. Tabnabbing: A new type of phishing attack. `http://www.azarask.in/blog/post/a-new-type-of-phishing-attack/`.

[15] Safe Browsing API – Google Developers. `https://developers.google.com/safe-browsing/`.

[16] D. R. Sandler and D. S. Wallach. <input type="password">must die! In *Proceedings of W2SP 2008: Web 2.0 Security & Privacy 2008*, Oakland, CA, May 2008.

[17] SiteKey Security from Bank of America. `https://www.bankofamerica.com/privacy/online-mobile-banking-privacy/sitekey.go`.

[18] StatCounter. Screen resolution alert for web developers.

[19] R. K. Suri, D. S. Tomar, and D. R. Sahu. An approach to perceive tabnabbing attack. In *Internation Journal of Scientific & Technology Research*, volume 1, 2012.

[20] S. Unlu and K. Bicakci. Notabnab: Protection against the "tabnabbing attack". In *eCrime Researchers Summit (eCrime), 2010*, pages 1 –5, oct. 2010.

[21] Z. Weinberg, E. Y. Chen, P. R. Jayaraman, and C. Jackson. I still know what you visited last summer: Leaking browsing history via user interaction and side channel attacks. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, SP '11, pages 147–161, 2011.

[22] L. Wenyin, G. Huang, L. Xiaoyue, Z. Min, and X. Deng. Detection of phishing webpages based on visual similarity. In *Special interest tracks and posters of the 14th international conference on World Wide Web*, WWW '05, pages 1060–1061, New York, NY, USA, 2005. ACM.

[23] M. Wu, R. C. Miller, and S. L. Garfinkel. Do security toolbars actually prevent phishing attacks? In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, CHI '06, pages 601–610, New York, NY, USA, 2006. ACM.

[24] Y. Zhang, J. I. Hong, and L. F. Cranor. Cantina: a content-based approach to detecting phishing web sites. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 639–648, New York, NY, USA, 2007. ACM.