The Salty Dogs
Christopher Grasing and Kimberly Allan
CSE 361 Web Security
Due Date: 12/3/2017

Shark Detector: A Tabnabbing Detection Chrome Extension

Introduction

Tabnabbing is an attack that takes advantage of human inability to keep track of many things at once. All the attacker has to do it have a script which detects loss of focus for some certain amount of time, and then transforms the page into what looks like a login page for something such as gmail. It's very simple yet effective if you have a bunch of tabs open.

We developed a Chrome extension for detecting these attacks. The basic idea is to monitor all the tabs when they are open, and compare how much they've changed when switched back to them. The extension can then warn you before you type in your precious information so a victim might investigate before entering their precious information.

Design Decisions

Professor Nikiforakis provided the basic idea for the design of this extension which was to take screenshots at intervals for use when comparing the images later on if focus is lost. Using this basic idea we knew the core of the extension would be a background page with a map of tabs to the latest screenshots. We did this by mapping screenshots to tab IDs which we could retrieve if a tab was switched back to.

Events obviously played a key part in our design as most of the functionality revolves around the user flipping around tabs. At the most basic level the extension just needed to watch for when the active tab changed and when the active tab's url changes to a new site or page. Filter out all URLs that aren't HTTP or HTTPS and you have the basic skeleton for the proper time to take screenshots and add things to the map.

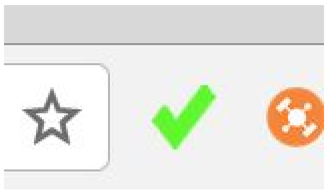
For the actual task of taking screenshots we knew we just needed to take an initial screenshot and spawn a setInterval thread to take more while the current page was the focus. Comparing these screenshots to the one taken when we switched back would use Resemble.js Library which provides the percentage changed as well as a diff image that highlights changes.

Initially we thought an overlay or a new tab would be the best way of showing this diff image but we decided on using a color-coded pageAction button that would show a popup of what changed. This allowed for a clean and separated way of viewing the changes which also had the added benefit of being less intrusive on sites which triggered false positives.

Tiers of Safety

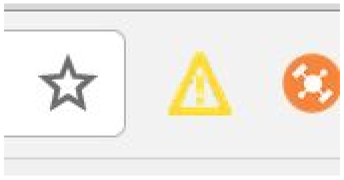
An integral part of our design is allowing the user to see the probability that their page was victim to a tabnabbing attack. For each tab open to a web page, a Chrome page action icon is displayed whenever a tab is made active upon re-navigating to the tab (the page was already opened, the user lost focus on the tab, and then the user refocuses on the tab). The safety margins are dependent on the percentage the visible area of the tab changed since the last visit to the tab. The following icons are possibly shown:

Green OK icon
Safety margin 1: 0 - 20% change



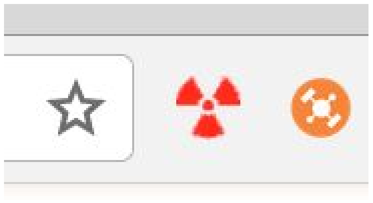
This safety margin indicates the smallest probability that this is a tabnabber tab. Some text might have changed in dynamic content or an advertisement might have changed. Most websites have this amount of advertisements.

Yellow Warning icon
Safety margin 2: 20 - 50% change



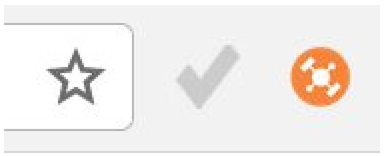
This safety margin indicates that this is a possible tabnabber tab. Decent portions of the page have changed. This may have been triggered by an advertisement change. Some websites have this amount of advertisement change, but login pages that are very sparse could also trigger this if the original page is sparse as well.

Red Hazardous icon
Safety margin 3: > 50% change



This safety margin indicates the largest possibility that this is a tabnabber tab. This may have been triggered, however, by a video stream, a delay in the rendering of page content, large portions of dynamic advertisements, or scrolling large amounts between screenshots. This is not realistic unless something suspect is going on, unless you are watching a large video player this means that you should be careful on this site.

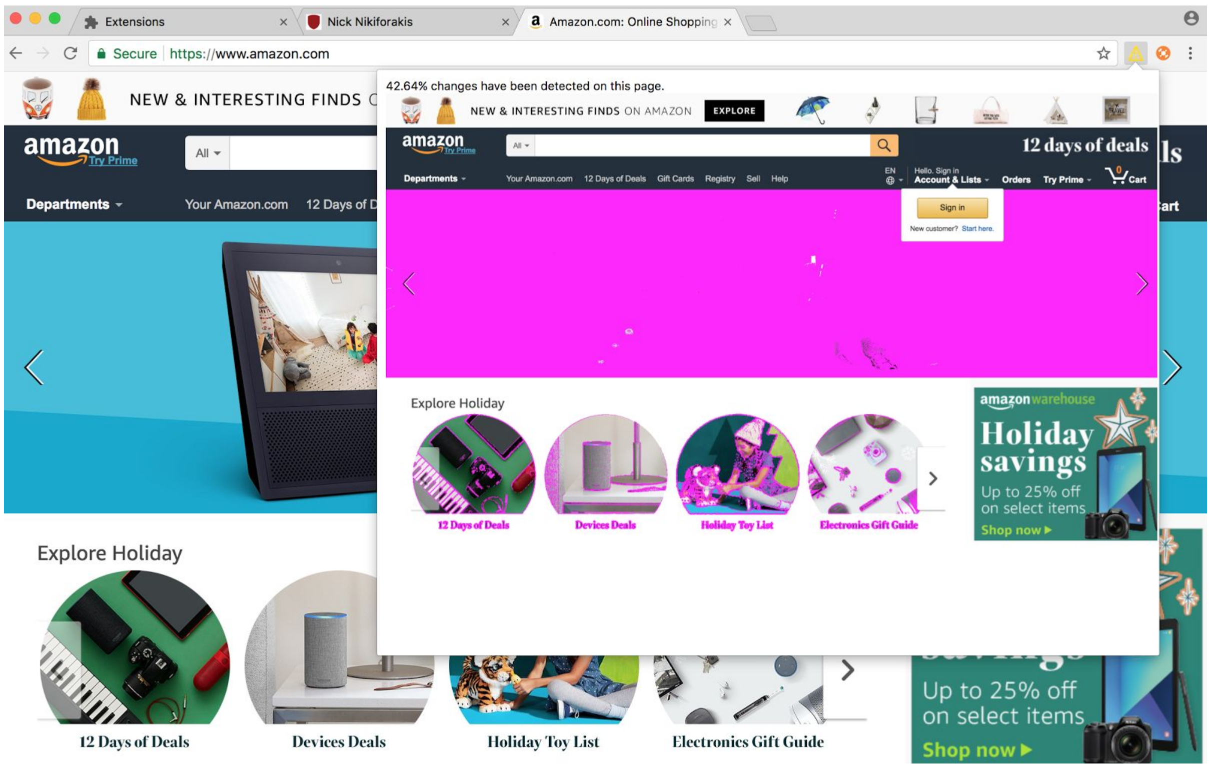
Disabled
Processing image, not comparing yet



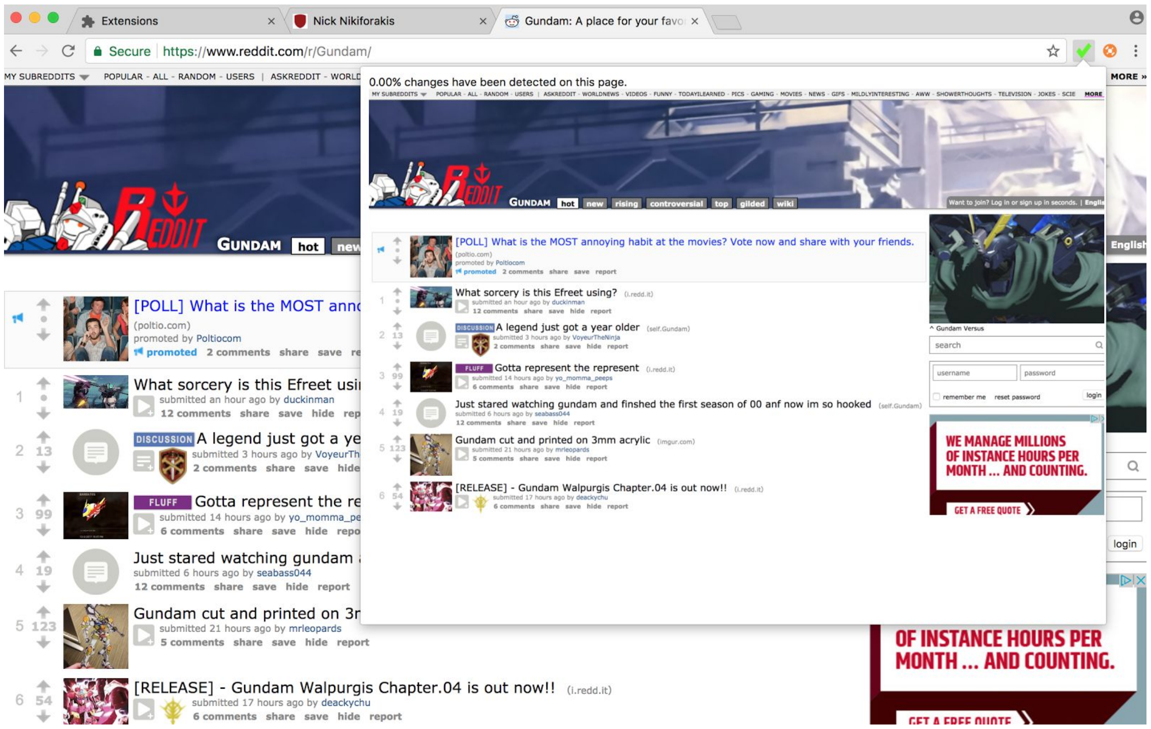
In the event that a tab has just been made or a new site has been navigated to there is no comparison so be done, a greyed out check is shown. If a tab has been switched back to there may be some milliseconds before the comparison is finished and this will be shown until the processing is finished.

The Extension In Action

It falls on the user of browser to determine whether or not that this browser was victim to a tabnabbing attack or one of the exceptions noted above. When the user clicks on the page action icon, a popup appears notifying the user of how much the page has changed since the last visit to this tab and the diff image generated from the last picture taken before the user navigated off the tab and the first picture taken upon re-navigating to the tab.


42.64% changes have been detected on this page.

An example of popup contents is shown above. The differences are highlighted in pink and on top, the percentage change in the page is displayed. Although this page falls into the yellow warning margin, a user looking at this page should be able to tell that the warning for this page happened due to a change in an advertisement banner and not a tabnabbing attack. It is important the that user looks at the URL to validate that this is indeed Amazon.com. By allowing the user to see the actual changes, false positives are less of a problem.



Extensions
Nick Nikiforakis
Gundam: A place for your favo
Secure https://www.reddit.com/r/Gundam/

0.00% changes have been detected on this page.

[POLL] What is the MOST annoying habit at the movies? Vote now and share with your friends.
(poltio.com)
promoted by Poltiocom
promoted 2 comments share save report

1 What sorcery is this Efreet using? (i.redd.it)
submitted an hour ago by duckinman
12 comments share save hide report

2 13 DISCUSSION A legend just got a year older (self.Gundam)
submitted 3 hours ago by VoyeurTheNinja
2 comments share save hide report

3 99 FLUFF Gotta represent the represent (i.redd.it)
submitted 14 hours ago by yo_momma_peeps
6 comments share save hide report

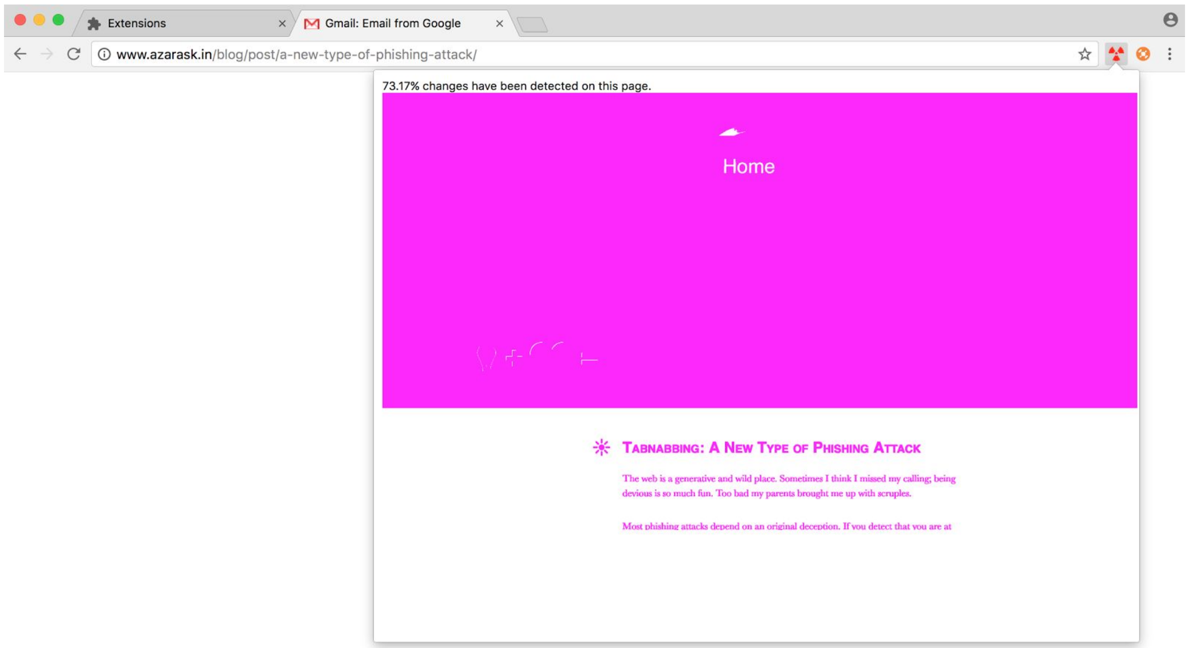
4 19 Just stared watching gundam and finshed the first season of 00 anf now im so hooked (self.Gundam)
submitted 6 hours ago by seabass044
12 comments share save hide report

5 123 Gundam cut and printed on 3mm acrylic (imgur.com)
submitted 21 hours ago by mrleopards
5 comments share save hide report

6 54 [RELEASE] - Gundam Walpurgis Chapter.04 is out now!! (i.redd.it)
submitted 17 hours ago by deackychu
6 comments share save hide report

WE MANAGE MILLIONS OF INSTANCE HOURS PER MONTH ... AND COUNTING.
GET A FREE QUOTE

This site was fairly static and didn't change at all after switching focus. So after the image is processed the green is shown and the user can still see the old screenshot of the page.



73.17% changes have been detected on this page.

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

This site simulates the attack (http://www.azarask.in/blog/post/a-new-type-of-phishing-attack/), it’s broken right now and displays a blank page but the change is still rather large. If it actually still had a gmail login it would have an even bigger change percentage.

Implementation

Libraries

The Shark Detector Chrome extension was implemented with three of chrome’s APIs (chrome.tabs, chrome.windows, and chrome.pageAction) and a Javascript library called Resemble.js that is able to compare the differences between two PNG images. The structure of the extensions will be discussed in this section.

chrome.tabs API - (https://developer.chrome.com/extensions/tabs)
chrome.windows API - (https://developer.chrome.com/extensions/windows)
chrome.pageAction API - (https://developer.chrome.com/extensions/pageAction)
Resemble.js - (https://github.com/Huddle/Resemble.js)

╬ Shark Detector Organization and Configuration

```
{
      "manifest_version": 2,
      "name": "shark-detector",
      "version": "0.1",

      "permissions": [
            "tabs",
            "activeTab",
            "<all_urls>"
      ],

      "page_action": {
            "default_icon": {
                  "24": "icons/green-okay.png"
            },
            "default_title": "shark-detector",
            "default_popup": "attack-detected.html"
      },
      "background": {
            "persistent": true,
            "Scripts":["resemble.js","constants.js","shark-utils.js",
                        "shark-detector.js"]
      },

      "author": "Chris Grasing & Kim Allan"
}
```
Figure 1: Manifest.json for Chrome extension configuration

The manifest page shown in Fig. 1 was used to configure the extension. The Chrome extension was made with background pages configured as the following…

```
      "background": {
            "persistent": true,
            "Scripts":["resemble.js","constants.js","shark-utils.js",
                        "shark-detector.js"]
      }
```

A Chrome background page is an HTML page that lasts for the lifetime that the Chrome extension is enabled. A Chrome background page was used rather than Chrome event pages since the solution design required the maintenance of global variables and Javascript timing events (setTimeout, setInterval, and clearInterval). Chrome event pages thus would not be viable with the solution design. The background page is indicated by "persistent" : true. The background HTML page was injected with the following scripts:

resemble.js: The Resemble.js library provides functions to compare two PNG images and find differences in structure and color. The screenshots taken with the chrome.tabs API of a single tab were compared with this library to detect changes in the tab's view over time and after change of focus events on the tab. The library also allows for the generation of an image that is the difference of two compared images. This "diff" image is displayed to users of the Shark Detector each time a tab is refocused/reactivated in order to show the user the differences between the page shown now and the last time on which the tab was focused.

constants.js: The global constants are defined here for the Shark Detector. These include a debug constant, time intervals and delays for the Javascript timing events, and icon paths for the Chrome page action icons for each tab.

shark-utils.js: This file defines all of the functions used to make the Shark Detector extension functional. There is a screenshot function used by setInterval, and functions to take screenshots and update the tab dictionary according to the necessary logic of the Shark Detector.

shark-detector.js: This file defines window and tab listeners to detect events such as tabs.onActivated, tabs.onUpdated, tabs.onRemoved, windows.onRemoved, and windows.onFocusChanged and global variables. These functions associated with these listeners call functions defined in shark-utils.js.

Chrome page actions are used by Shark Detector to display an icon for the currently active tab and show a popup containing the diff images generated for each tab.

```
      "page_action": {
            "default_icon": {
                  "24": "icons/green-okay.png"
            },
            "default_title": "shark-detector",
            "default_popup": "attack-detected.html"
      }
```

Whenever an image comparison is made for a tab (when a user navigates back to a tab previously created), the page action icon is set with one of 3 icons to notify the user of the Shark Detector's determination of how probable a tabnabbing event occurred (green for least likely, yellow for more likely, and red for highest probability), the HTML for the page action popup is set with the percentage difference of the tab images captured and the diff image generated, and the page action is made visible.

```
<HTML>
      <body>
            <div style="width:800px;height:600px;">
                  <label id="msg"></label>
                  <img style="MAX-WIDTH:100%; MAX-HEIGHT:100%;"
            id="inserter" alt="loading...">
                  <script src="attack-detected.js"></script>
            </div>
      </body>
</HTML>
```

Figure 2: attack-detected.html

The attack-detected.js simply inserts the percentage difference and the diff image source.

The permissions given to the Chrome extension were dependent on the Chrome API functions and tab properties used, including a function that captured the visible tab and the tab's URL property.

```
"permissions": [
      "tabs",
      "activeTab",
      "<all_urls>"
]
```

The "<all_urls>" permission is necessary for chrome.tabs.captureVisibleTab that takes pictures of the tab currently open. The "tabs" permission allows for access to the Tab.url property. This property is used to check whether or not the URL uses HTTP or HTTPS since we are not keeping track of any non-Web based tabs. While these permissions are very open and give the extension access to much of the user's browsing data, no data is used maliciously and no page content is being modified or communicated with when the extension is active. However the idea act of taking screenshot itself means by definition some level of intrusiveness.

╬ Shark Detector Logic

The extension keeps track of the following globals:

image_dict: dictionary of the tab ID to an array containing the tab's URL and data URL of the last screenshot taken of the tab when it was the active tab
screenshot_thread: variable holding the return value of the setInterval function for the screenshot function to be called every 5 seconds. The screenshot function calls chrome.tabs.captureVisibleTab on the currently active tab, and stores the PNG as a data URL in image_dict.
delay_thread: variable holding the return value of setTimeout function. This registers a function that checks whether or not a new or updated dictionary entry needs to be made for a tab (due to a new tab or new URL in a tab) or if a tab was re-navigated to, requiring an image capture and comparison.
current_img: after an image comparison, the diff image is stored here so that the page action popup can access the diff image
current_pct: after an image comparison, the percentage difference is stored here so that the page action popup can access the percentage difference.

The logic for the Shark detection is defined as follows:

chrome.tabs.onActivated:

When a tab is activated by creating a new tab…

1. Clear all timers already set by another tab (in screenshot_thread and delay_thread)
2. Check if the URL is defined and is HTTP or HTTPS
3. Set a small timeout to allow most pages render
4. Screen capture the visible tab
5. Add a new dictionary entry for this tab with the recently captured image
6. setInterval on the screen capture function (which takes a screenshot and overwrites the dictionary entry for this tab) every 5 seconds.

When a tab is activated by navigating to a tab that already exists…

1. Clear all timers already set by another tab
2. Check if the URL is defined and is HTTP or HTTPS
3. Set a small timeout to allow most pages render
4. Screen capture the visible tab
5. Compare the previous image entry in the dictionary with the current image using Resemble.js
6. Check the mismatch percentage and choose the corresponding icon
7. Set globals with the diff image data URL and the percentage difference to be rendered by the popup html
8. Set the icon of the page action and make the page action visible
9. Update the dictionary with the most recent captured image
10. setInterval on the screen capture function every 5 seconds.

chrome.tabs.onUpdated:

Perform the same steps as those for when a tab is activated by creating a new tab, except only when the tab is the currently active tab. Also, when checking if the page is HTTP or HTTPS, if it is not, then delete the tab entry from the dictionary if it already exists.

chrome.tabs.onRemoved:

When a tab is removed, delete the tab from the dictionary. Only clear the timers (screenshot_thread and delay_thread) if the removed tab is the currently active tab.

chrome.windows.onFocusChanged:

When there is a focus change on a Chrome window, clear all timers and check if the window now focused on is a Chrome window. If the newly focused window is a Chrome window, query for the currently focused window's active tab and perform the same steps as when a tab is activated by navigating to an already existing tab.

chrome.windows.onRemoved:

When the window close button is pressed, clear all timers. The chrome.tabs.onRemoved event will be fired for all of the tabs closed in the process.

Split Of Work

The scope of the project was not too large overall so in reality we ended up both working on every aspect of the project. However in the beginning we planned to split the work up by having Chris working on the events being fired to correctly track when the user's tab needs to be captured. Kim worked on taking the actual screenshots and using resemble.js to check the image diffs and working on window events. As we progressed, testing and finding edge cases, as well as dealing with the async issues because we both weren't JavaScript web devs became the main bulk of the work we did. Research into how Chrome works was done by both of us throughout the entire project and we often were programing on each others computers on test branches that did not make it to the final master branch.

Overall we definitely both learned a lot from this project. It was surprising to see how easy it is to develop a browser extension, and also at how powerful and invasive they can be made to be. Users should be careful of what extensions they download and what permissions they have.