Raymond Lam - 109834504
Nauman Shahzad - 109813732
CSE 361 - Project 2
Project Report

Introduction

One of the many types of phishing attacks is tabnabbing: an attempt to exploit a user's sense of security after they have navigated to a webpage. The presumed habit is that users are trained to earnestly inspect the aspects of a page for maliciousness only when they first land on it; they are assumed to drop their guard if they do not perceive any initial evil intent.

With tabnabbing, an attacker must first get a user to navigate to their page. This page is seemingly benign at first. It can resemble any harmless page: blogs, forums, etc. are all possibilities. After a certain amount of time, the page changes its appearance to that of a significant, substantial site (ex: Paypal, Yahoo) to pose as it. If the user has been focused on another tab, then when they return to the tab containing the attacker's site, they will see the login page of this site and login, thinking that Paypal, Yahoo, etc. have logged them out by accident. Because the user is no longer cautious (since there was no initial evil intent), there is now a higher chance to steal their credentials.

The attacker is banking on the user not being more aware: if the user sees that a page that previously lead to a forum has turned into a login form for Gmail in their absence, all they need do is check the link in the address bar. "super-benign-blog.com" obviously would not ask for Paypal login credentials, so the user can still save themselves if they catch this. But what about those less astute? Is there any way for them to protect themselves?

Our extension is an attempt to solve this problem. By taking pictures of each tab periodically, we can compare a tab from when before it lost focus to when it regained focus to ensure that it has not changed dramatically, then alert the user to the possibility of an attack.

Design Decisions

We decided to make this extension for Chrome because of its prevalence. According to statcounter.com, it had approximately 55% of the world's browser market share in the month of November 2017 (followed by Safari with about 15%, UC Browser with about 8%, and Firefox with about 6%), so an extension in Chrome would affect the most people.

In order to compare the before and after images of a tab, we used the suggested resemble.js library. It was easy to use and allowed us to skip having to create a comparison method ourselves.

We used a linked list data structure to store the before images of each tab. Each node in this linked list contained three values: the before image (in data url format), the url of the tab, and the link to the next node.

Initially, we followed the suggestion of splitting the before and after screenshots into squares and comparing each square. However, we noticed that resemble.js, in its callback after comparison, contains fields misMatchPercentage and getImageDataUrl(), which contain the percentage difference between the before and after and the image with the colored differences, respectively. Because of this, we opted to not use the suggestion and used these values directly instead.

Color coding the extension icon based on the changes seemed to be a reticent way of alerting the user of possible changes to the page (since the user might not be paying attention to the address bar and in turn, the extension icon), so we elected to notify the user of the misMathPercentage with an alert. We also decided to compare the before and after links of the favicons of each page since changes to them may not be easily noticeable.

Implementation

The extension is run entirely from its background page. The background utilizes three JavaScript files: resemble.js, LinkedList.js, and background.js. resemble.js is used to compare the before and after images, LinkedList.js is used to store the before images, and background.js is a driver for LinkedList.js.

Background.js contains all functionality to listen for the necessary events. It uses chrome.tabs.captureVisibleTab to capture the tab in focus and installs a listener on chrome.tabs.onActivated to call captureVisibleTab. This allows the extension to screenshot each tab whenever it gains focus. Whenever a tab gains focus, a screenshot of it at that moment is taken and the linked list is updated. An alarm is also set up to update the image of a tab if the user spends a lot of time on the same tab via the chrome.alarms set of functions.

LinkedList.js stores not only the before images but also the urls for each image to discern which tab is which. It is also here where resemble.js is called to compare the images. Assuming the mismatch percentage is greater than zero resemble.js returns an image with all the mismatches highlighted. Otherwise, there was no change in the page so the user is not alerted for anything.

We then take this image and inject it as an iframe into the user's current visible tab via chrome.tabs.executeScript function. This iframe has an opacity of 0.5 so that the user can still navigate the page. The iframe is also fixed in place at the y offset of the tab when it regains focus; it does not follow the user if they scroll to a different part of the page. This was accomplished by setting the iframe's position to absolute and setting its top to the y offset. We set the iframe's pointerEvents attribute to none so that the user could still interact with the page

even if the iframe was overlaid on top of it. Lastly, its z-index was set to a high value to ensure that it always appeared above the tab's content. The user is also alerted as to what the exact mismatch percentage was.

Resemble.js contains the methods needed to compare the before and after images. They were written by Huddle at https://github.com/Huddle/Resemble.js/.

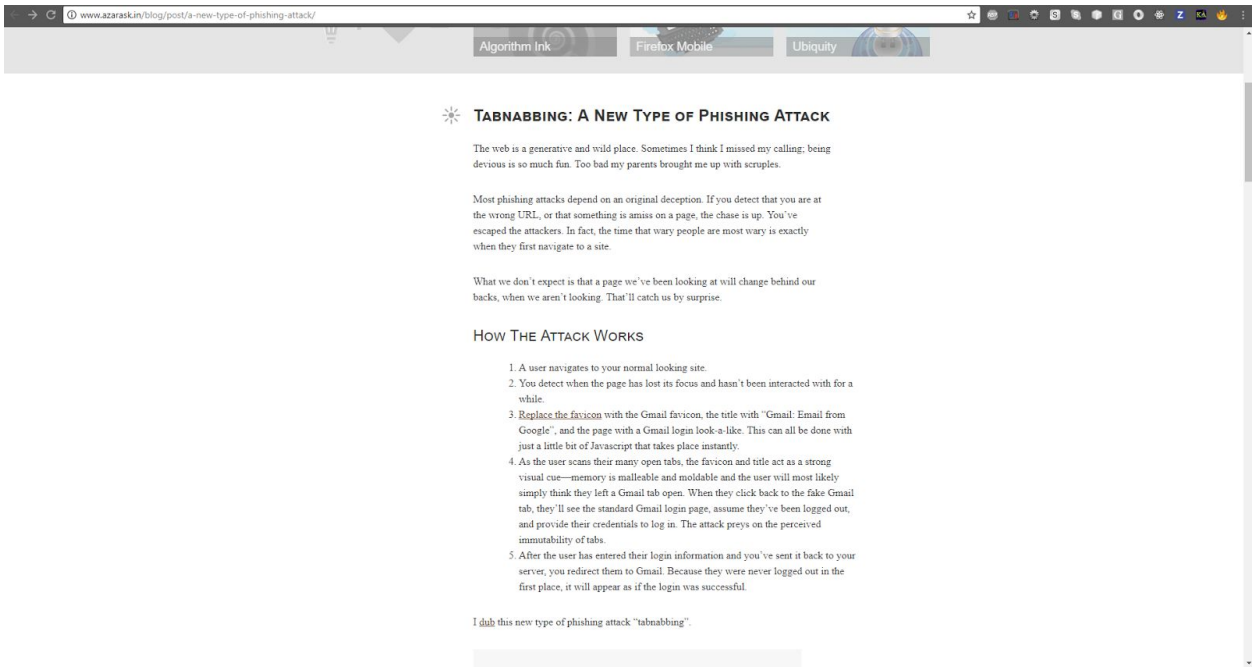
Installation

Customize Chrome -> More tools -> Extensions -> Load Unpacked Extension. Navigate to the directory which contains the extension, then click OK.

Sample Output

The images were taken from this website. The site provides a demonstration of a site drastically changing its appearance after a user tabs out for a little bit.

Before:



After(without highlighting):

After(with highlighting):

Resemble.js returns the differences as a highlight of pink. We saw no need to change this color and left it as is.

Shortcomings

Chrome's API for capturing a tab is limited in its capabilities in that it is only able to capture the current visible tab. There was not a way to capture one last image of the tab the user clicked out of. This means that if a user scrolled on a page and clicked out of the tab before the alarm for updating the image went off then when they return to that page there'll be a false positive of page having "changed", because the image wasn't completely up to date.

There were also numerous times when Chrome's tab API would fail to capture the current visible tab. If it failed, it would log a console error only indicating that it failed to capture anything This occurs when clicking into a tab, it does not always happen but it occurs frequently enough to take note of. This made testing whether or not the extensions worked a lot longer than if Chrome's API had not failed to capture an image. Because of this, we had to first check if both the before and after images were not null before comparing them.

On pages like google hangouts that already had an iframe in the page, our injected iframe with the highlighting gets pushed to the bottom of the page rather than overlaid right on top of where the visible mismatches were. In the case of hangouts a user would have to scroll all the way down to see image of where the mismatches were.

Sometimes, the iframe did not seem to overlay the iframe over the page evenly on some of the machines we tested it on. We could not figure out why this was the case.

Responsibilities

Nauman: I worked mainly on the development of the Chrome Extension. I designed and implemented our Linked List application, in which each node has a field for the before image and the url with which that image corresponds to. If the extension attempted to add another node with the same url it'd take that current image and compare it with the before image and then inject the iframe with the highlighted differences if the mismatch percentage was greater than twenty percent. I set up the chrome alarms to have the extension periodically update the image for the current visible tab, which occurs once every minute. Also, I incorporated the resemble.js library manually because chrome extensions do not support node js so we could not utilize npm to install the package for us.

Raymond: I worked on the Chrome extension, specifically following the initial suggested guidelines in the project description. After setting up the Bitbucket repository and creating the skeleton of the extension (manifest.json, icon.png, background.js), I figured out how to screenshot the active tab and split said screenshot into squares/rectangles using canvas. However, we decided to scrap that idea in favor of using just the misMatchPercentage value that resemble.js returns. I then worked on comparing the before and after links of the favicons to ensure they were not changed as well as fixing the injected iframe into place and testing resemble.js's functionality. I later wrote up the report.

Conclusion

Our chrome extension is aimed towards being able to assist with tab phishing detection in taking images of the currently visible tab and comparing how the current image differs if at all from before. It also checks to see if the favicon also changed on the tab and alerts the user if it did. In an attempt to reduce the number of false positives in mismatches, we set the threshold to be at least a twenty percent mismatch before we alert the user to any differences and highlight them. Although this may not be a perfect or thorough solution to pruning out every false positive it does help to reduce them on messaging sites like facebook's messenger.com.

Much more can be done to upgrade the extension's functionality, but the basic defense is there. Many of the problems we encountered while developing this extension may have been solvable in other ways, and the problems we were not able to solve may have had solutions we could not realize.