CSE 331 Project Report

by Lily Zhong, Jessica Lin, Patrick Chan, Cynthia Cheung

Summary

Tabnabbing is a form of phishing that takes advantage of the lack of focus on a particular website when a user is on another tab. In this scenario, when the user first opens a particular website, they will be aware of what website they are opening. However, as the user opens more and more tabs during their browsing session, they eventually lose track of the different websites they have opened. In such a case, a malicious website can take advantage of the window's lack of focus on the particular website. The attacker can completely change the appearance of the targeted website and create a false login, which may appear to be a well known website's (Google, Facebook, Amazon, etc.). As the user isn't likely to inspect the browser's URL, they may just assume that Facebook/Gmail has logged them out and wants the user to re-login. Thus, the user simply logs back in and as a result their credentials were stolen by the attacker.

In this project, we are trying to prevent such an attack. We created a browser extension that alerts users when a tabnabbing attack occurs. When the user first opens a webpage, screenshots are being taken on regular intervals of the web page as the user browses it. Each time a new screenshot is taken, the new screenshot replaces the previous screenshot kept in the storage. When the user opens a new tab and exits out of the current webpage into the new webpage, a loss of focus is detected. After the user returns to the previous tab, a new screenshot is taken and the two screenshots are compared. If there are any changes on the page, the changes are highlighted in red and the user is notified. If the user believes that a malicious attack has occurred, they have an option to add the URL to a tabnabbing blacklist.

As an extra precaution, when the user opens a webpage, the webpage's URL is checked against a server that contains a list of blacklisted websites. If the URL is found on the list of blacklisted websites, the user is notified of potential tabnabbing attacks that can occur on the particular webpage.

Instructions

Server

1. Ensure that you have the latest version of Golang installed; instructions can be found here.

2. Clone the GitHub repository or download it as an archive.
3. Inside a terminal, navigate to the directory where the repository is located.
4. Navigate to the server directory inside the repository using `cd team-cia/server`.
5. Compile the server executable using `go build`.
6. Run the server using the command `./server &`.

Extension

Prerequisites...

Have a working computer and have both a Chrominum-based browser and Git installed.

1. To load up the extension, you can clone our GitHub repository.
2. Go to chrome://extensions and click “Load unpacked”.
3. Select the `extension` folder and load it into the browser.
4. Be sure to restart your browser in order for the extension to take full effect.
5. Visit stoptabnabbing.online/dev_console to add domains to the blacklist. There will be some preset ones for testing purposes.
6. Continue browsing.
    a. If a website is not on the blacklist, and it redirects you while you’re on another tab, a notification will occur (depending on your OS, it will either appear in the top right corner or the bottom right corner). This notification will ask you whether you want to see the differences between the latest screen capture it took before you left the tab and the latest screen capture it took when you came back to that tab.
        i. The first image shown is the *before* image.
        ii. The second image may show blobs of red, as it is supposed to show the differences between the *before* image and the *after* image. The *after* image is not shown.
        iii. You will be prompted whether you want to add that domain to the blacklist.
    b. If a website is on the blacklist, it will prompt you whether you want to leave the website (as it’s known to be malicious) or you want to stay.
    c. If the latest screen capture of a tab when you left is not the same size as the latest screen capture of a tab when you came back, the comparison will not work. Our extension unfortunately does not handle different sized images.
    d. If a website is not on the blacklist, and it does not redirect you while you’re browsing, nothing fun will happen.

Architectural Design

Server

We chose to use MongoDB to store the blacklist where each entry contains the timestamp and the hostname. To make the blacklist more effective, we chose to use the hostname instead of the full URL since there would be portions that are unique to each user in many cases. If we had stored the full URL, it would only blacklist that one webpage, which is not desired. The server listens for requests to either add/delete a URL from the blacklist or to retrieve the blacklist.

Extension

NAIVE IMPLEMENTATION

For event handling, we decided that it made the most sense to utilize Chrome API's event handlers for when a tab is newly activated. This triggers the `onActivated` event handler in the API. If we used this method, then one way to capture the most naive form of tabnabbing would be to screenshot the webpage upon tab activation and the tab is fully loaded, which would then trigger the `onUpdated` event handler, and then screenshotting it again when the user comes back. This would mean that we would have to keep track of the capture upon the first activation and the capture when they come back.

The blur handler came in when we decided to deviate from the naive approach. Since the built in blur event handler in JavaScript works in the DOM specifically, we kept that in `content.js`, apart from `background.js`. Instead of capturing just one photo upon tab activation, we could utilize `blur` to capture the last thing that the user saw right before they exited.

CLIENT-SIDE PROCESSING AND SERVER-SIDE PROCESSING

The captures have to be stored somewhere and then processed using a tool that compares two input images and outputs an image of the differences. The best way to store images would be to store them locally, since Chrome API allows us to use unlimited browser storage (and if unspecified, a maximum of 5MB).

Here is a compilation of reasons we decided to work with image processing on the client-side.

Client	Server
+ Unlimited storage using Chrome API's `unlimitedStorage` permission	- Served as a black box since user does not know what happens behind the scenes within a server
+ Local storage means that in case of a data leak from the server, the screenshots associated with the user will remain safe	+ Better performance
	+ Storage is not a problem

The data could have been saved and processed through a server too, but since the server is regarded as a black box to the user, the user may be suspicious of what happens behind the scenes. There are some pros to storing it client-side.Client-side storage is local, so in case of a data leak, the screenshots associated with the user will remain safe.

SETTING THE INTERVAL

To improve things even more (and thereby making it even more dynamic), the extension would take multiple captures within a short period of time. For example, instead of just taking one when a user activates the tab or when they leave that tab (to navigate to another), the extension would capture the visible area in the active tab every *x* milliseconds. By doing so, there would be more captures to ensure security while the user navigates in the webpage; the latest capture would be likely be the most updated one.

COMMUNICATION BETWEEN DOM AND CHROME API

But `content.js` and `background.js` serve different purposes! How will background.js, the one that's utilizing all of Chrome API's functions, know when a blur event, which is notified in content.js, has occurred so that it knows to take a screen capture? We used Chrome API's way to communicate messages, called "message passing". Instead of a one-time message, however, we created a long-lived connection using 2 ports that listened to messages within both files. This allowed us to send repeated messages back and forth rather than send a single message.

IMAGE PROCESSING AND NOTIFYING THE USER

We decided to go with three different kinds of notifications/alerts: if a difference is detected, if the images sizes are different, and if the user enters a malicious site. Instead

of directly showing the images as a popup, we decided to give users the option whether or not they want to view the difference for a particular site. We decided to use pixelmatch instead of resemble.js because it was more lightweight and faster. We found that placing it into our background.js file rather than importing it made modifications easier. Since pixelmatch is unable to compare images of different dimensions, the user should be informed when no comparisons are made. If the user changes the size of their window before returning back to a tab, it would be important to inform them that no comparisons will be made, rather than have them continue browsing a possibly malicious site. Otherwise, the resize event handler will tell background.js that such event has occurred and the local storage for that tab should be updated accordingly. Lastly, an alert for if a user enters a malicious site was used to inform the user of the possible risks of entering that site.

The popup window displays the image prior to the user leaving the tab and the difference after coming back to the tab. Any difference will be in red. We show the before image so that the user has a reminder of what the site was like before leaving the tab. We don't show the image after coming back because it's redundant information when the difference is also shown. Any red in the difference image indicates a change between the site before leaving the tab and when the user comes back. We also included the site's URL so that the user will know which site the comparison was made from. We decided to make it up to the user to determine whether or not to add that particular site to the blacklist.

The popup window will always display the latest comparison, regardless of whether the user leaves the window open, or doesn't click the notification. If the user leaves the window open, the contents will update as the user enters and leaves tabs. We decided to do this so that the user won't be left with a list of comparisons from multiple sites. In addition, no stacking of notifications will appear either, since the operating system doesn't support stacking of notifications.

Who did what?

Patrick Chan
I was primarily responsible for the server component, which kept a database of URLs that were decided to be malicious by the users of the extension. The blacklist could then be shared with all the users to improve the amount of protection that the extension can provide. I also took part in debugging the extension as we were having a lot of issues with notifications showing up randomly and integrating the server blacklist with the extension.

Lily Zhong

I worked on the client side using primarily JavaScript. Since our extension aimed to resolve tabnabbing at a very basic level, I set up the focus, blur, and resize event handlers that would trigger when a user focuses on the DOM, clicks outside of the DOM, and resizes the browser window. I also implemented it so that it would capture a new image every X milliseconds so that the browser's storage would keep track of what the webpage looks like every now and then and debugged the extension along the way.

Jessica Lin

I worked on the client side, focusing on the image comparison section of the extension using the pixelmatch image comparison library. Since pixelmatch is unable to compare images of different dimensions, and the dimensions were necessary to create the output image of the differences, I took a snippet of code from stackoverflow to obtain the width and height of both images. Once pixelmatch output the difference, it was then displayed as a popup window that would only display the latest comparison. The window will contain the image of the site before leaving the tab, and the differences after re-entering. I also took part in the testing of the extension for any possible bugs.

Cynthia Cheung

I worked on the client side. I used primarily JavaScript and Chrome's API to set up notifications and alerts that will alert users of any malicious attacks. One such notification occurs when the user enters back into a tab that previously lost focus and changes in the two different screenshots are detected. The notification gives the user an option to view the changes or to ignore the changes. Another notification occurs when the user opens a blacklisted website. I also took part in creating the css for styling of the html for the popup window.