A Hybrid Approach to Detect Tabnabbing Attacks

by

Hana Sadat Fahim Hashemi

A thesis submitted to the
School of Computing
in conformity with the requirements for
the degree of Master of Science

Queen's University
Kingston, Ontario, Canada
August 2014

Abstract

Phishing is one of the most prevalent types of modern attacks, costing significant financial losses to enterprises and users each day. Despite the emergence of various anti-phishing tools and techniques, not only there has been a dramatic increase in the number of phishing attacks but also more sophisticated forms of these attacks have come into existence. One of the most complicated and deceptive forms of phishing attacks is the tabnabbing attack. This newly discovered threat takes advantage of the user's trust and inattention to the open tabs in the browser and changes the appearance of an already open malicious page to the appearance of a trusted website that demands confidential information from the user. As one might imagine, the tabnabbing attack mechanism makes it quite probable for even an attentive user to be lured into revealing his or her confidential information. Few tabnabbing detection and prevention techniques have been proposed thus far. The majority of these techniques block scripts that are susceptible to perform malicious actions or violate the browser security policy. However, most of these techniques cannot effectively prevent the other variant of the tabnabbing attack that is launched without the use of scripts.

In this thesis, we propose a hybrid tabnabbing detection approach with the aim of overcoming the shortcomings of the existing anti-tabnabbing approaches and techniques. Our approach combines five heuristic-based metrics with data mining techniques to keep track of the major changes made to the structure of a webpage whenever a tab loses its focus. We develop our approach as a browser extension for Mozilla Firefox and evaluate its effectiveness and performance using a dataset consisting of legitimate and tabnabbing websites. Our evaluation results convey a significant improvement over the existing techniques, indicating that our approach can be utilized as a viable means for protecting users from tabnabbing attacks.

Acknowledgments

I would like to extend by most sincere gratitude to the following people who have made the completion of this thesis possible:

My supervisor, Dr. Mohammad Zulkernine for his guidance, encouragement, and true support during my M.Sc. degree program;

Dr. Komminist Weldemariam, for his helpful insights and comments on my work;

My family, for their unconditional love and support, and the faith they have always had in me through every step of my life;

My husband, for his heartfelt support and much needed inspiration;

And my colleagues at the Queen's Reliable Software Technology Group (QRST), for their moral support and the enjoyable time I spent with them during my M.Sc. degree program.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

The Web has become an indispensable global platform as it glues together various services as a one stop shop (*e.g.*, daily communication, sharing, trading, and collaboration). Users of the Web often store and manage critical information that attracts attackers who misuse the Web and its infrastructure (*e.g.,* browsers) to exploit vulnerabilities for underground economy. One of the most common types of contemporary attacks is phishing. In this attack, fraudulent websites mimic the appearance of legitimate websites to trick users into revealing confidential information and credentials. Phishing attackers often target well-known and trusted websites and employ simple modifications on them in order to make the attack more believable. The stolen information is then used to make illegal economic profit (*e.g.,* making online banking transactions, purchasing goods, etc.) [1, 2].

In recent years, phishing has become a chronic cyber-security challenge, victimizing the ever-growing online population of the world and posing a serious threat to the Internet economy. According to the Anti-Phishing Working Group (APWG) [3], "*There were at least 115,565 unique phishing attacks worldwide in the second half of 2013. This is nearly a 60% increase over the 72,758 seen in the first half of 2013.*" This places phishing attacks among the most prominent types of Internet attacks.

Typically, a phishing attack leverages social engineering techniques by sending a large number of luring emails containing a link — usually an obfuscated URL that is a concealed malicious URL imitating a legitimate URL — that can potentially redirect users to an attacker-controlled website. These emails seem to be sent, for instance, from the user's bank or any popular website as the link mimics the look-and-feel of a trusted website. Upon a successful redirection to the attacker-controlled website, the attacker can automatically collect sensitive information (*e.g.,* bank account numbers, credit card numbers, and passwords) and use it for underground economy [1, 4, 5, 6].

While still struggling with the destructive effects of typical phishing attacks, Web users were introduced to a new and more threatening form of phishing named tab-nabbing [7]. The classical phishing attack relies on the deception of users with a similar URL and/or bogus resemblance to the original website, whereas tabnabbing takes advantage of the user's inattention to the open tabs in the browser [8]. Tabn-abbing takes over the inactive browser tabs by changing the appearance of an open page in a tab to the appearance of a trusted website. In this attack, a malicious but legitimate-looking webpage that is already open in one of the tabs reloads its contents to resemble a trusted website demanding certain information from the user. In this way, the unsuspecting user most likely falls into the attackers' trap and inputs con-fidential information into the malicious webpage. The collected information is then sent to the attacker-controlled server and can be used for various types of abuses.

Two variants of the tabnabbing attack have been identified to date: the script-based variant [7] and the script-free variant [9]. While the first variant relies on the execution of scripts, the second one launches an attack without the use of scripts and based on page refreshes that occur in certain time intervals predetermined by the attacker. A script is a sequence of commands that is used to automate the execution of certain tasks [10]. In this context, by script-based attack, we mean an attack that is launched by exploiting JavaScript. JavaScript is a scripting language (*i.e.,* a programming language that supports scripts). The script-based variant of the tabnabbing attack can be tackled using some of the existing script-blocking browser extensions (See [11, 12, 13] for examples). The script-free variant, however, is more challenging to prevent, and the existing script-blocking techniques cannot guarantee full defense and immunity against it. For the time being, the two abovementioned variants are the only proof of concepts available for the tabnabbing attack.

1.2 Overview

With the aim of addressing the above challenge, in this thesis, we present a hybrid tabnabbing detection approach, named **TabsGuard**. TabsGuard combines heuristic-based metrics and data mining techniques to detect tabnabbing attacks. To be able to flag a webpage as tabnabbing or benign, our approach keeps track of the changes made to the structure of a page during the time the page is not interacted with. For this purpose, we introduce five heuristic-based metrics and use them to measure the degree of changes made to the tree representation of each webpage whenever a tab loses focus. We then leverage data mining techniques to analyze the occurred changes and determine whether the webpage is tabnabbing or benign. In case the attack is detected on a browser, we use our prevention mechanism to inform the user with an alert message, suggest the user to add the page to a local blacklist on her browser, and redirect her to a safe domain.

In an attempt to address the shortage of tabnabbing pages we have at our disposal, we generate a number of tabnabbing pages that we use for the experimental evaluation of TabsGuard. We also study the existing tabnabbing detection and prevention techniques and identify certain criteria for categorizing these techniques. These criteria can be used to provide a more vivid image of the strengths and shortcomings of the existing anti-tabnabbing approaches and techniques.

We develop our proposed tabnabbing detection approach as a browser extension for Mozilla Firefox and evaluate it using the top 1,000 legitimate websites from Alexa [14] and 11 tabnabbing pages that we generated. Our evaluation of TabsGuard consists of evaluating its capability of distinguishing between tabnabbing and legitimate pages as well as evaluating its performance as a browser extension. We also compare TabsGuard with an existing tabnabbing detection technique (**TabShots** [8]), demonstrating that our approach can outperform TabShots when the right data mining technique is used for analyzing the changes made to a webpage.

1.3 Contributions

The major contributions of this thesis are as follows:

- We identify five heuristic-based metrics that we use to determine the degree of changes made to the overall structure of a webpage. We select these metrics based on the assumption that they reflect inherent characteristics of a webpage and correlate with its structure.

- We propose **TabsGuard**, a hybrid approach to detect tabnabbing attacks and develop it as an extension for the popular Mozilla Firefox browser. In particular, we leverage the five identified heuristic-based metrics in addition to seven different anomaly detection techniques (*e.g., k*-NN [15], Local Outlier Factor [16], Connectivity-Based Outlier Factor [17], Histogram-Based Outlier Score [18], etc.) to detect tabnabbing attacks. Unlike some of the existing tabnabbing detection techniques, our approach is capable of detecting both script-based and script-free variants of the tabnabbing attack. We also evaluate the effectiveness and performance of TabsGuard as a browser extension.

1.4 Thesis Organization

The remainder of this thesis is organized as follows. We proceed by introducing phishing and tabnabbing attacks in detail and provide background information on these attacks (Chapter 2). We review the existing phishing and tabnabbing detection approaches and techniques proposed so far and point out their strengths and weaknesses in Chapter 3. Chapter 4 describes our proposed approach for detecting and preventing tabnabbing attacks. The experimental evaluation of our approach and its comparison with the existing anti-tabnabbing techniques are presented in Chapter 5. In Chapter 6, we conclude our work and outline directions for future work.

Chapter 2

Background

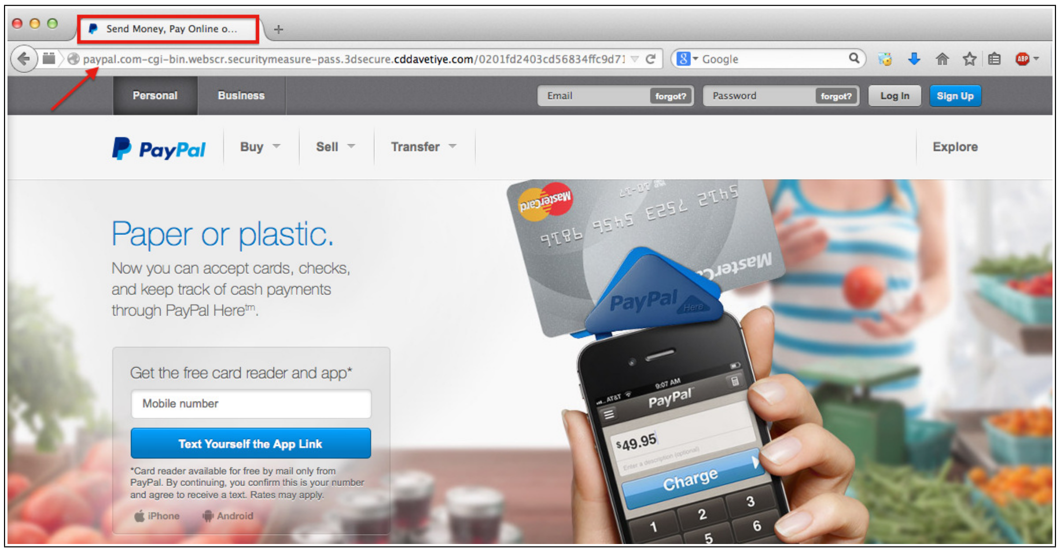
In this chapter, we present background knowledge on phishing and tabnabbing attacks. In addition to providing detailed definitions and introducing the terminology in this area, this chapter aims at highlighting the seriousness of phishing and tabnabbing attacks and the necessity of preventing them before irreversible damages are incurred. We first describe the phishing attack and the threats it poses to the Web and mobile users (Section 2.1). Then, we describe the detailed scenario of the tabnabbing attack along with the methods that can make this attack more threatening (Section 2.2). Finally, we call attention to the existing challenges for detecting phishing and tabnabbing attacks (Section 2.3).

2.1 Overview of the Phishing Attack

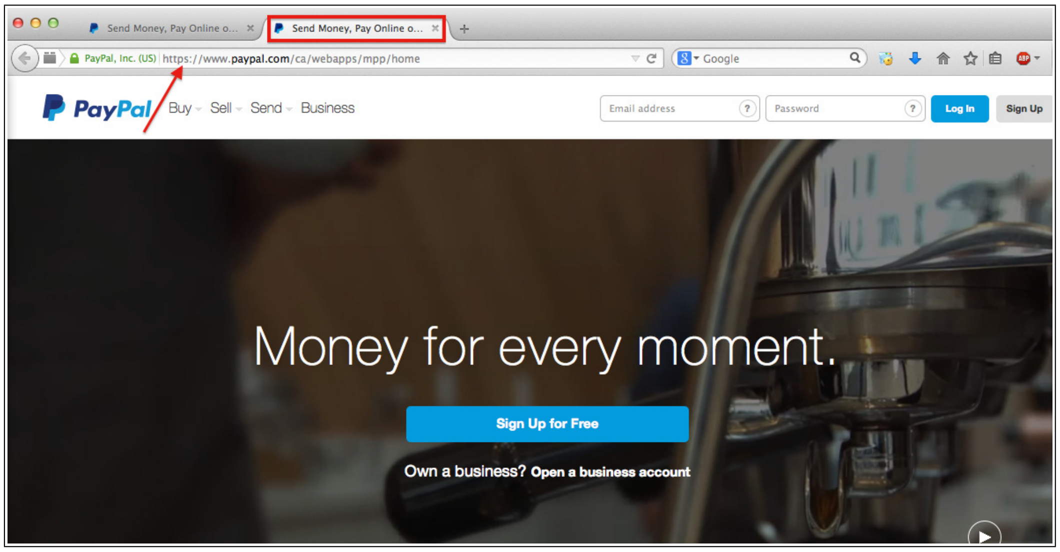
Phishing is an online fraudulent attempt made with the purpose of identity theft. Phishing attackers apply both social engineering and technical deception to steal users' confidential information for future gains. A victim of phishing attack is redirected to a malicious website by unknowingly clicking on an obfuscated link that is usually sent to her in an email message. A phishing website resembles its legitimate counterpart except for some minor differences that are not usually noticed by the user. When the unsuspecting user tries to access a trusted website, she is ultimately redirected to a fake website and tricked into submitting personal information or credentials [1, 4].

Social engineering — the art of manipulating people into acting in a certain way or giving away confidential information [19] — is an important feature of phishing attacks. The outstanding aspects of social engineering employed in phishing attacks are persuasion and generation of certain feelings such as threat, concern, or urgency in targeted victims. For instance, words such as *"risk"*, *"suspended"*, and *"identify"* are very common to be found in phishing email messages [20]. Moreover, if the victim receives the email message bearing malicious content when she is stressed, natural skepticism is much lower, and she will more easily fall into the attackers trap.

For the general user, well-known banks, Internet Service Providers (ISPs), or social networking websites are the places in which phishing attacks are more likely to occur. Figure 2.1 shows an example of a phishing website. In Figure 2.1-(a), a phishing website mimicking Paypal, and in Figure 2.1-(b) the original Paypal website are illustrated. The phishing website looks quite convincing, bearing the same title and Paypal logo as the original Paypal website as well as having a similar login form to that of the original website. The major difference between the two websites that might be overlooked by the user is their URL. However, the word "Paypal" can still be spotted in both URLs which complicates the identification of the phishing website. Therefore, the inattentive user most likely falls prey to the attack and enters her email address and password to the fake login form.


Send Money, Pay Online o...
paypal.com-cgi-bin.webscr.securitymeasure-pass.3dsecure.cddavetiye.com/0201fd2403cd56834ffc9d7
Personal
Business
Email
Password
Log In
Sign Up
PayPal
Buy
Sell
Transfer
Explore
Paper or plastic.
Now you can accept cards, checks, and keep track of cash payments through PayPal Here™.
Get the free card reader and app*
Mobile number
Text Yourself the App Link
*Card reader available for free by mail only from PayPal. By continuing, you confirm this is your number and agree to receive a text. Rates may apply.
iPhone
Android
$49.95
Charge

(a): Phishing Paypal website


Send Money, Pay Online o...
Send Money, Pay Online o...
PayPal, Inc. (US) https://www.paypal.com/ca/webapps/mpp/home
PayPal
Buy
Sell
Send
Business
Email address
Password
Log In
Sign Up
Money for every moment.
Sign Up for Free
Own a business? Open a business account

(b): Original Paypal website

Figure 2.1: An example of a phishing website.

According to Orman [19], a typical phishing attack consists of a number of the following elements:

a. an email message that looks like ordinary messages of a trusted organization;

b. a list containing the email addresses of intended victims;

c. a website with a convincing name that mimics a trusted website;

d. an IP address for the website's server;

e. malware that collects victims' information;

f. malware that victims might be persuaded to download and install on their computers.

In addition to the more common form of phishing which uses email as a means for making the attack, there are some other forms of phishing attacks attempted using Short Message Service (SMS) or phone calls. In 2006, SMS was identified as a new means for phishing attacks, introducing SMS phishing or SmiShing to the Web users. In this form of phishing, an SMS from an apparently reputable organization or financial institution asks for the receiver's immediate attention and prompts her to call a telephone number or visit a website where the attacker steals the victim's confidential information [21].

Another form of phishing which can be more difficult to detect is spearphishing — a phishing attack that targets a specific organization. This type of attack appears to come from a trusted source that is well-known to the victim (*e.g.,* his employer), and might contain personal information, such as the victim's correct full name. As soon as the victim exposes his confidential information, he opens a gateway for the attacker to access the internal network of the organization, make monetary transfers, or obtain valuable information [19].

Trying to address the increasing emergence of different forms of phishing attacks, researchers have come up with several different types of techniques to detect and prevent phishing attacks. Some examples are blacklisting, whitelisting, webpage analysis, and use of heuristics, machine-learning classifiers, or search engines. In Chapter 3, we will discuss these techniques in extent.

2.2 Tabnabbing Attack

Tabnabbing is a computer exploit and a complex form of phishing attack that takes advantage of users' trust to the open tabs in the browser. The idea behind this attack is that users are the most cautious the first time they navigate to a webpage, because they do not expect the page to change when it is out of focus. Tabnabbing takes over the inactive browser tabs by changing their title, favicon, and contents to those of a trusted website. Favicon is the small icon associated with a website or webpage that is displayed on the browser tab [22] (Marked as 1 in Figure 2.2). The title of each webpage is also displayed on its corresponding tab in the browser right next to the favicon (Marked as 2 in Figure 2.2).

The tabnabbing attacker gains access to one of the open tabs of the user's browser when the user downloads a third-party script such as a Flash widget. Third-party scripts are software components that can be reused for creating highly distributed web applications for advertising purposes, tracking visitor statistics, and widgets [23, 24]. A widget is a stand-alone application with limited functionality that can be embedded into third-party websites. A Flash widget is a widget developed using Adobe Flash-based technologies [25]. To fall prey to the tabnabbing attack, the user is persuaded to download a malicious program disguised as a third-party script (*e.g.,* a Flash widget) in the first place. Using the malicious program, the attacker will be

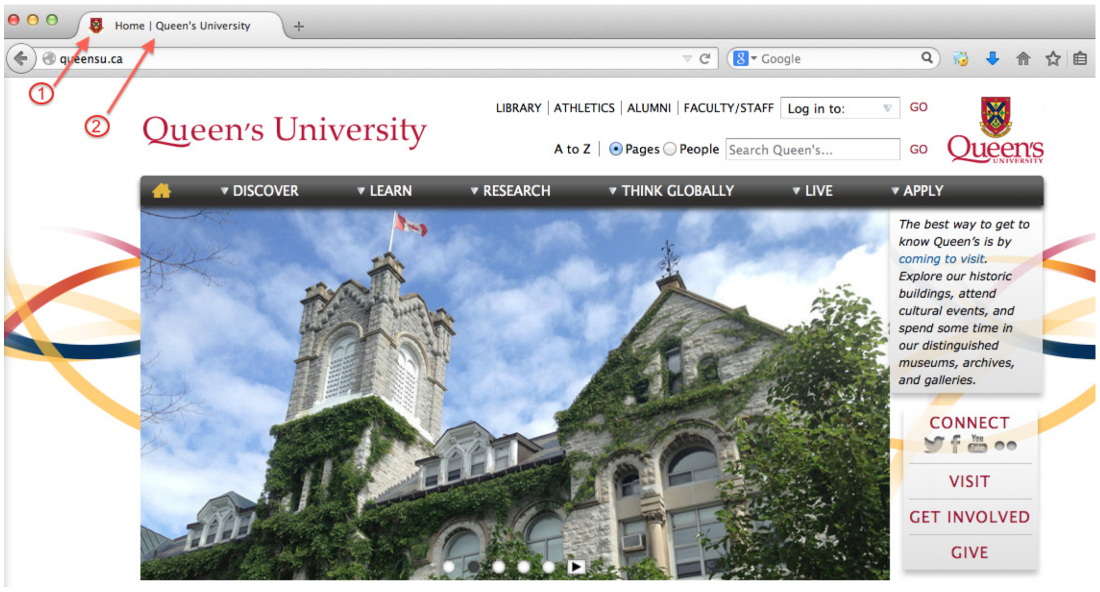

Home | Queen's University
queensu.ca
1
2
Queen's University
LIBRARY | ATHLETICS | ALUMNI | FACULTY/STAFF
Log in to:
GO
A to Z | Pages People
Search Queen's...
DISCOVER
LEARN
RESEARCH
THINK GLOBALLY
LIVE
APPLY
The best way to get to know Queen's is by coming to visit. Explore our historic buildings, attend cultural events, and spend some time in our distinguished museums, archives, and galleries.
CONNECT
VISIT
GET INVOLVED
GIVE

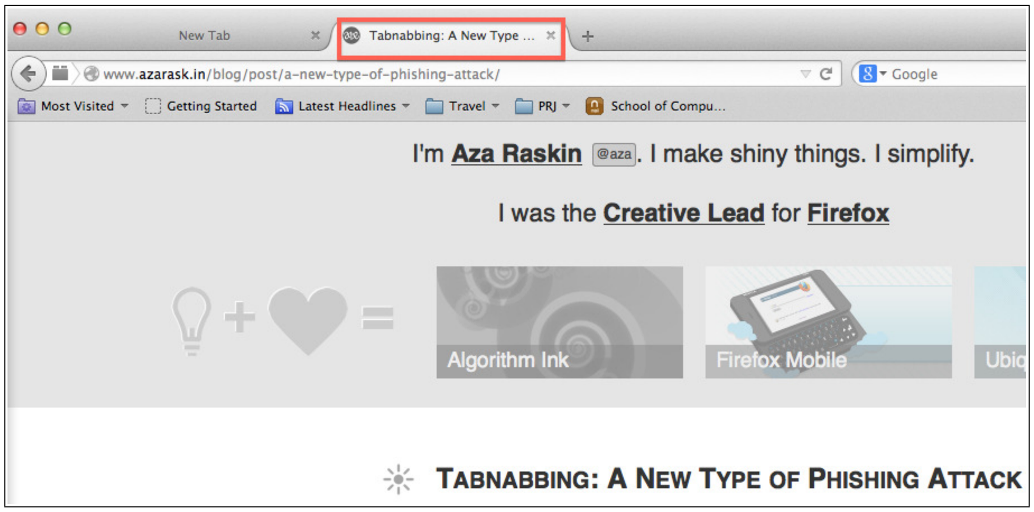
Figure 2.2: An example of title and favicon in a website.

able to control an open tab in the user's browser when the page is not interacted with by its user for some time. Thus, every time the user includes a third-party script or a Flash widget in a webpage, she makes herself more susceptible to this kind of attack.

The tabnabbing attack was first identified by Aza Raskin in 2010 [7]. Figure 2.3 shows an overview of the tabnabbing attack in which a malicious but legitimate-looking page (Figure 2.3-(a)) changes to the appearance of a trusted website (Figure 2.3-(b)) when the corresponding tab is inactive. The general attack scenario as proposed by Aza Raskin assumes that the user has a number of open tabs in her browser. The steps for this scenario are as follows:

1. The user opens an already malicious webpage in a tab, navigates away from the tab after a short while, and moves to the other open tabs.

2. The malicious page detects that the tab is out of focus. It reloads the title, favicon, and contents of the page in the tab, replacing the title and favicon with

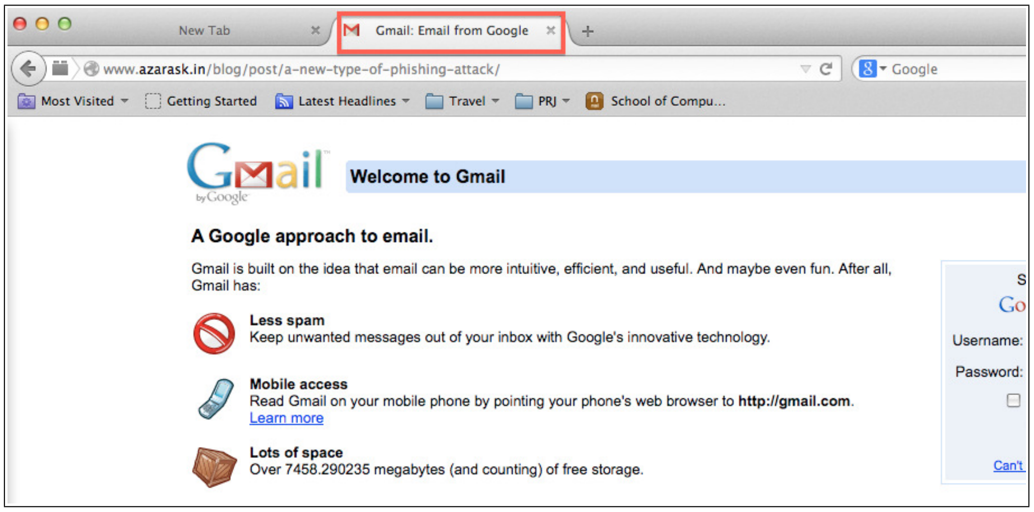


I'm **Aza Raskin** @aza. I make shiny things. I simplify.

I was the **Creative Lead** for **Firefox**

TABNABBING: A NEW TYPE OF PHISHING ATTACK

(a): Before tab switch



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.

(b): After tab switch

Figure 2.3: Overview of the tabnabbing attack.

those of a trusted website and the appearance of the page with one looking alike the appearance of the trusted website (in this case, the login page of Gmail).

3. The user moves back to the tab, views a trusted website's login page, assumes that she has been signed out, and enters her credentials into the login form.

4. The user's personal information is sent to the attacker-controlled server.

5. The malicious webpage redirects the user to the trusted website — of which the user has never been signed out — to make it seem like the login attempt was successful [7].

Some alterations have been made to the above attack model by Lokesh Singh [26]. In this alternative scenario for the tabnabbing attack, iframes are used for launching the tabnabbing attack. It should be noted that an `<iframe>` is an HTML element used for embedding a web document within the current document [27]. Moreover, *timers* with predefined values replace the tab switch events that trigger the attack in the original attack scenario.

Using the following complex attack methods, the tabnabbing attack could be developed into more threatening versions. Some attacks make use of Cascading Style Sheets (CSS) history miner to access the user's full browsing history or find out which websites the user visits on a regular basis[1]. History miners use malicious code to check whether a user visits certain websites [28]. Therefore, they can be used to make the tabnabbing attack more plausible by targeting the wesbites that the user has visited in the past or has them opened in other tabs. Other types of the attack check whether the user is currently logged in to the target website and perform the attack when the user is actually logged in so as to make the attack more convincing. Cross-site scripting (XSS) vulnerabilities [29, 30] can also be utilized to force the attack to be launched by other websites. XSS vulnerabilites enable attackers to inject malicious scripts into benign websites. These vulnerabilities are found in web applications that fail to validate or encode user input before returning it to the client-side web

[1]This attempt is no longer possible in Firefox betas.

browser. Furthermore, to make the tabnabbing attack more convincing, the attacker can replace the contents of the tab under attack with a page informing the user that her session has timed out and she needs to reauthenticate. This is common to happen on bank websites [3].

A much more dangerous version of the attack can be imagined if the URL of the malicious tab reloads too and is replaced with a legitimate-looking URL. This can happen in case look-alike Unicode Domain Names are used [31]. The Unicode Standard [32] is a character coding system that supports displaying of the written texts of diverse languages. Unicode provides a unique number for every character regardless of the platform, program, or language. The use of Unicode in domain names makes Web forgery easier due to the existence of similar visual representations of Internationalized Domain Name (IDN) strings in the browser. IDNs contain non-ASCII characters and are written in languages that may or may not use the Latin alphabet. Therefore, it would be almost impossible for the user to distinguish between the original URL of a website and a very similar URL using a non-Latin alphabet.

There are two proof of concepts available for the tabnabbing attack. The first one [7] is based on the execution of JavaScript. Here, the state of a tab (*i.e.,* active or inactive) is identified by JavaScript `onblur` and `onfocus` events. The second proof of concept [9] does not rely on the execution of scripts and uses the meta-refresh HTML attribute. This attribute causes the automatic reloading of the page after a predefined time interval. Unlike the script-based variant of the attack, the script-free variant does not directly depend on user activity, but on the given time interval by which the attacker hopes the user has switched tabs and thus is not attentive to the tab under attack. Compared to the large number of existing anti-phishing solutions, there are very few techniques specific to tabnabbing detection and prevention. We will discuss these techniques in Chapter 3.

2.3 Phishing and Tabnabbing Detection

One popular solution to protect users from phishing and its variants is to enhance Web browsers with additional security features so that users can be warned whenever a phishing website is being accessed. Such browser security is often provided by a mechanism known as blacklisting, which is the most common technique to defend against phishing attacks [2]. However, some attackers use a type of evasion called IP cloaking to check incoming connections and compare them against an anti-phisher list [19]. Using IP cloaking, if the incoming connection is from a phishing detection website, the suspected website is identified as benign. Another issue that complicates the detection of phishing websites is that their uptime is quite short. In fact, the average phishing website is active for a few days, some even for only a few hours.

The effectiveness of the existing phishing (or tabnabbing) detection solutions is evaluated by having them identify known phishing and legitimate websites, and measuring two metrics known as false positives and false negatives. When testing known, already reported phishing websites, false negatives exist if the phishing website is identified as a legitimate website. When testing legitimate websites, false positives exist if the website is identified as a phishing website, but is in fact legitimate. The evaluation results of the existing anti-phishing solutions are reported in terms of false positives and false negatives. A solution with a lower false positive and false negative rate detects phishing attacks more accurately; hence, one can more confidently rely on its detection. The two aforementioned metrics are also helpful in comparing the performance of different phishing detection techniques with each other.

2.4 Summary

In this chapter, we described phishing and tabnabbing attacks and brought attention to the vitality of proposing more efficient approaches for detecting such attacks. First, we described the phishing attack mechanism, presented the basic elements a phishing attack consists of, and introduced some of the new forms of this attack. Then, we first explained how a tabnabbing attack could happen and presented the typical scenario of the tabnabbing attack as first proposed by Aza Raskin. We then described the existing technological methods that make tabnabbing attacks almost undetectable by the user as well as the two variants of the tabnabbing attack. Finally, we described the existing challenges of detecting phishing and tabnabbing attacks and introduced the expressions used when addressing and comparing phishing and tabnabbing detection techniques.

Chapter 3

Related Work

In this chapter, we present the existing anti-phishing and anti-tabnabbing approaches and techniques and highlight their strengths and weaknesses. Section 3.1 provides an overview of the most common types of techniques that are used for detecting phishing attacks. Section 3.2 details the state-of-the-art phishing detection and prevention approaches. Section 3.3 elaborates on the existing techniques for tabnabbing detection and prevention and introduces the criteria we propose for comparing these techniques.

3.1 Overview of Anti-Phishing Approaches and Techniques

Several anti-phishing approaches, techniques, and tools have been proposed in the literature. Some of these solutions focus on list-based approaches (*i.e.,* blacklists [2, 33] or whitelists [4]) to distinguish between phishing and legitimate websites. There are some techniques that leverage the combination of heuristics and machine-learning techniques [2, 4, 33] to check one or more characteristics of a website to detect phishing attacks. Other approaches leverage the properties of the webpage and utilize search engines [5, 34] or data mining tools [20, 35] to analyze suspicious pages according to certain properties derived from HTML code, URL, and contents.

In this section, we present a categorization of the existing anti-phishing solutions and describe some of the state-of-the-art in this area. In general, the existing anti-phishing solutions can be categorized as follows with respect to the approach they use.

1. **Blacklist-based Techniques:** Blacklists are online repositories of known phishing websites. Blacklist-based techniques match a given URL with a list of URLs belonging to a blacklist. These techniques detect phishing websites by either doing a database look-up or conducting a dynamic and predictive detection. Database look-up is performed before navigating to a website by having the browser query its database to determine whether the requested URL is already verified as phishing or not. A popular blacklist used in almost every blacklist-based technique is PhishTank [36]. PhishTank is an online freely accessible repository that provides the URLs of verified phishing websites to developers.

   Although blacklists provide simplicity in design and ease of implementation, a major problem with them is their dependence on a complete database containing all known phishing websites. Blacklist-based techniques are unable to detect zero-day phishing attacks (*i.e.,* new, unreported attacks) and spearphishing attacks.

2. **Whitelist-based Techniques:** Whitelists are lists containing the URLs of a number of trusted websites that a user accesses on a regular basis. Trusted websites can be added to the whitelist by the user herself. Whitelist-based approaches depend on whitelists to protect users from phishing attacks. Each time the user attempts to navigate to a website that is not within the whitelist, she is either denied access to the website or asked to add the website to the whitelist. Every website whose URL does not exist in the whitelist is identified

as a suspected website. Therefore, for the whitelist to work properly, it should ideally contain every legitimate website, which is not possible. This is the reason why whitelist-based techniques normally lead to a large number of false positives.

3. **Heuristic-based Techniques:** Heuristic-based techniques check one or more characteristics of a website to detect phishing attacks. These characteristics can be the URL of the suspected website, its HTML source code, or its contents. Presence of external links to the target website to get images and other content in real-time, as well as the existence of dots (.) and IP addresses in the URL are also heuristics that can be used to identify phishing websites [4, 33]. After determining the heuristic characteristics, most heuristic-based approaches leverage machine-learning algorithms for making a classification of the suspected website. The main strength of such approaches is their ability to detect zero-day phishing attacks [37].

4. **Analysis-based Techniques:** Webpage analysis techniques analyze a downloaded webpage and determine whether it is phishing or legitimate by examining certain properties of the page derived from its HTML source code and its URL.

List-based techniques benefit from simplicity and speed as in most cases they mainly do a database look-up. However, their major weakness is that they strongly rely on the completeness of the database in order to work properly. Heuristic-based techniques on the other hand do not suffer from this drawback and are able to detect zero-day phishing attacks. Nevertheless, heuristic-based techniques are more prone to result in a false alarm. In addition, heuristic-based techniques enjoy shorter response times, but get less accurate results compared to list-based techniques.

3.2 State-of-the-Art Anti-Phishing Approaches

In this section, we briefly describe a number of the existing approaches proposed for defending against phishing attacks.

Prakash *et al.* [2] propose PhishNet, an approach that takes advantage of both heuristic-based and blacklist-based techniques to detect phishing attacks. They propose five heuristics to generate new URLs by considering simple combinations of known phishing websites. The authors also develop a matching algorithm to match a URL's components with existing URLs in the blacklist based on piece-wise similarity. Having low false negative and false positive rates is the significant strength of their approach. Additionally, they believe applying more heuristics could further improve their results.

Belabed *et al.* [4] present a hybrid approach for detecting phishing attacks. Their hybrid approach combines personalized whitelisting and machine-learning techniques. Their proposed whitelist is an XML file consisting of the login pages of the top 10 most attacked websites and serves as a filter for the next phase. Thus, phishing pages that are not blocked by the whitelist go through a Support Vector Machine (SVM) classifier, which classifies websites according to eight features. The proposed approach has a relatively high false positive rate which the authors associate with using the SVM classifier.

Dunlop *et al.* [5] make use of images to do a content-based analysis on suspected websites. They develop a toolbar, named GoldPhish, which (1) uses Optical Character Recognition (OCR) to read text from the images of a webpage; (2) feeds the extracted text line by line to a search engine; (3) grabs top-ranked search results; and (4) compares the top 4 search results with the website under test. The major benefit of this technique is that GoldPhish takes the hidden text in images and website logos into account, providing more hints to label a website as phishing or legitimate, hence achieving zero false positive rate. However, GoldPhish is limited by the amount and style of text, logos, and images caught in the OCR image, and it delays the rendering of a webpage due to the OCR procedure and the exhaustive search engine look-up.

The research conducted by Bin *et al.* [1] is used for effectively protecting card numbers and passwords of online bank users. In the proposed approach, important information of banks (*e.g.,* the bank's name, the IP address of the bank's DNS server, and the range of bank's card numbers) are stored in a database. When a bank's card number is detected to be sent to a suspicious website, an inverse DNS query is sent to the DNS server of the bank to verify whether the suspicious website is in fact a phishing website or not. This approach is implemented as an Internet Explorer Toolbar and claimed to be accurate in terms of resulting in a low false negative rate.

Addressing the problem of the short uptime of phishing websites, Gupta and Pieprzyk [33] propose a hybrid model that converges blacklisting, heuristic-based techniques, and the vote of moderators of a phishing detection community such as PhishTank about a website being phishing or legitimate. The authors attempt to reduce the number of verifications required to be done by the moderators by processing each URL redirection request through various stages. The final decision about the website being phishing or legitimate is made using majority voting between 3 or 5 moderators. As a result of applying this approach, the response time for detecting a website as phishing or legitimate is decreased. However, the credibility of this approach is dependent on the reliability of the moderators who classify URLs as phishing or legitimate.

Zhang *et al.* [34] propose CANTINA, a content-based approach to detect phishing websites. CANTINA makes use of the Term Frequency-Inverse Document Frequency (TF-IDF) score of the terms on websites. TF-IDF is a weight used to determine the importance of a word to a document in a collection of documents [38]. After calculating the TF-IDF weight of each term on the webpage, CANTINA generates a lexical signature from the terms with highest values of TF-IDF. This signature is fed to Google search engine. In case the domain name of the suspected page matches with the domain name of the top 30 search results, the suspected page is detected as legitimate. Otherwise, it is detected as phishing. CANTINA also takes advantages of some heuristics such as the age of the domain name, and the existence of well-known logos, suspicious URLs and etc. CANTINA's evaluation results indicate achieving a very low false positive rate, which could be owing to CANTINA's use of numerous heuristics for URLs.

Ludl *et al.* [35] conduct an experimental study on the effectiveness of blacklist-based and analysis-based phishing detection solutions. To study the effectiveness of the blacklists provided by Microsoft and Google, the authors periodically check these blacklists for the existence of the most recent phishing websites reported by PhishTank. The authors also present a number of properties for a webpage with respect to its HTML source code and URL, and they evaluate the effectiveness of these properties using machine-learning techniques. Their study shows that blacklists can effectively detect phishing attacks, whereas the analysis-based technique that they use suffers from a high false positive rate.

Webber *et al.* [20] propose some phishing detection criteria and test the ability of several machine-learning classifiers to detect phishing emails. The authors employ a text analysis technique to find the most frequent words in a collection of phishing messages and build datasets with respect to 19 attributes. They achieve the most accurate detection results when using neural networks and decision trees as classifiers.

Last but not least, as part of the anti-phishing campaign, Mozilla has also introduced *Mozilla Persona* [39] to protect users from being tricked into entering their personal information into fake login forms. This solution serves as an authentication system in which email addresses are used as identifiers to eliminate the necessity of multiple username/password submissions for signing into various accounts. Thus, the burden of keeping users' identity, information, and credentials safe rests on the browser. This can reduce the risk of accidentally sending credentials to a malicious server; however, whether*Mozilla Persona* can effectively protect users against phishing (and tabnabbing) attacks is a controversial topic due to the security concerns this authentication system raises [40].

3.3 Existing Anti-Tabnabbing Techniques and Tools

In this section, we discuss the existing anti-tabnabbing solutions. We categorize these solutions into two groups. The first group includes script-blocking browser extensions that block scripts which are susceptive to perform malicious actions or violate the browser security policy (Section 3.3.1). The second group, on the other hand, includes those techniques that are specifically designed to detect or prevent tabnabbing attacks (Section 3.3.2). Table 3.1 provides an overview of the existing anti-tabnabbing solutions as well as our proposed approach, TabsGuard. This table helps us compare and contrast the existing tabnabbing detection and prevention tools and techniques with respect to the following criteria:

Table 3.1: Overview of the existing tabnabbing detection and prevention tools and techniques.

Criteria / **Techniques**	Use of Whitelists	Use of Blacklists	Browser	Script-Based Attack Prevention	Script-Free Attack Prevention	Technology in Use
NoScript [11]	Yes	No	Firefox	Yes	Yes (passive prevention)	JavaScript
Controle de Scripts [12]	No	No	Firefox	No	No	JavaScript
YesScript [41]	No	Yes	Firefox	No	No	JavaScript
NotScripts [42]	Yes	No	Chrome & Opera	Yes	No	JavaScript; HTML5 storage caching
ScriptSafe [43]	Yes	Yes	Chrome	Not by default	No	JavaScript
Script Defender [44]	Yes	No	Chrome & Opera	Yes	No	JavaScript
Script Block [13]	Yes	No	Chrome	Yes	No	JavaScript
TabShots [8]	No	No	Chrome	Yes (passive prevention)	Yes (passive prevention)	Screenshot comparison; Threshold value
NoTabNab [45]	No	No	Firefox	Yes (passive prevention)	Yes (passive prevention)	HTML layout; Topmost elements' positioning
Suri *et al.*'s Approach [46]	No	No	None	Yes (passive prevention)	No, only checks script tags	Extracting text between script tags; Rule-based system
Sarika & Paul's Approach [47, 48]	No	Yes	None	Yes (passive prevention)	Yes (passive prevention)	Visual changes; Multiagent systems
Our Approach: TabsGuard	No	Yes	Firefox	Yes (active prevention)	Yes (active prevention)	HTML DOM; Heuristics & data mining techniques

a. whether the tool/technique uses whitelists;

b. whether the tool/technique uses blacklists;

c. which browser has the tool/technique been deployed on (*i.e.,* as a browser extension);

d. whether the tool/technique prevents the script-based variant of the tabnabbing attack, and whether the prevention is passive (*i.e.,* merely displaying an alert message) or active (*i.e.,* warning the user and preventing the user from further proceeding);

e. whether the tool/technique prevents the script-free variant of the tabnabbing attack, and whether the prevention is passive (*i.e.,* merely displaying an alert message) or active (*i.e.,* warning the user and preventing the user from further proceeding;

f. the particular technology or idea the tool/technique uses to make tabnabbing detection feasible.

In what follows, we discuss the anti-tabnabbing solutions presented in Table 3.1 in more detail.

3.3.1 Script-Blocking Browser Extensions

The following browser extensions provide protection against the script-based variant of the tabnabbing attack — except for **NoScript** that provides a passive protection against the script-free variant of the attack as well. This protection is dependent on the default behavior of extensions towards preventing JavaScript code from execution on untrusted domains.

NoScript [11], **Controle de Scripts** [12], and **YesScript** [41] are Firefox-specific solutions. NoScript uses a whitelist-based approach, allowing JavaScript and other embedded content to be executed only by websites that the user trusts. Thus, the default behavior of NoScript is blocking all scripts that are not whitelisted, a behavior compromising the normal functioning of most websites. Additionally, being whitelist-based, it relies on the user's choice to allow scripts on a website, and therefore its tabnabbing protection mechanism is mainly manually provided. Controle de Scripts allows the user to control what JavaScript can do on the browser by adding extra settings to the browser preferences window, whereas YesScript lets the user make a blacklist of websites that are not allowed to run JavaScript. The main weakness of YesScript is its reliance on blacklists as blacklists always need to be up-to-date.

ScriptSafe [43] and **ScriptBlock** [13] are both Chrome-specific solutions. ScriptSafe provides some of NoScript's functionalities and uses both whitelist-based and blacklist-based approaches. This extension does not prevent the script-based tabnabbing attack by default. In our understanding, this might be due to its dependability on blacklists. ScriptBlock [49] implements a whitelist-based approach to control JavaScript, iframes, and plugins.

Both **NotScripts** [42] and **Script Defender** [44] are implemented as extension services for Chrome and Opera browsers. NotScripts provides a high degree of NoScript-like control for JavaScript, iframes, and plugins. Like NoScript, NotScripts implements a whitelist-based approach thereby leaving the decision to allow or block the scripts at the hands of the user. Moreover, it does not prevent inline scripts that are embedded in the HTML code of a webpage. On the other hand, Script Defender[1] allows JavaScript and other embedded content to be executed only on trusted domains that are whitelisted by the user. The default settings of Script Defender [50] block internal scripts and plugins. However, this extension allows external scripts, images, and iframes unless the preferences are changed by the user.

In general, there is a lot of criticism on this group of anti-tabnabbing solutions. For instance, being the most popular browser extension to overcome tabnabbing attacks, NoScript cannot *effectively* prevent the script-free variant of the attack. NoScript blocks page refreshes on unfocused tabs and instead lets page refreshes automatically execute only when the tab gets in focus again. However, the reloading of the page happens any way, and it is likely that it still does not get noticed by inattentive users the moment they navigate back to the tab. Furthermore, among the browser extensions mentioned above, NotScripts, ScriptSafe, Script Defender, and Script Block for Chrome provide no protection against the script-free variant of the tabnabbing attack. Similarly, YesScript and Controle de Scripts provide no protection against either variant of the attack. The advantage of TabsGuard over this group of anti-tabnabbing solutions is that our approach does not rely on the user's decision to stop malicious scripts from running. Instead, we detect the pages that are potentially malicious with a very high probability and warn the user of the threat.

3.3.2 Tabnabbing Detection and Prevention Techniques

To the best of our knowledge, there are only four approaches for tabnabbing detection and prevention; **TabShots** [8], **NoTabNab** [45], the approach proposed by Suri et

[1]This extension has recently been removed from Chrome and Opera repositories.

al. [46], and the approach proposed by Sarika and Paul [47].

TabShots [8] is a Chrome extension that records the favicon and a screenshot of a webpage once when it is visited for the first time and once after a tab switch event occurs. TabShots then compares the data obtained at these two times. For the screenshot comparison, De Ryck *et al.* divide the screenshots of the page to 10x10 tiles (*i.e.,* blocks) and compare each tile with its counterpart. If the observed changes are higher than a predefined threshold value, a visual overlay appears on the page to warn the user about the possibility of the tabnabbing attack. TabShots has been evaluated against the top 1,000 Alexa websites, showing that 78% of the websites go through less than 5% changes, 19% fall within 5%-40% changes, and 3% go through more than 40% changes. The main deficiency of Tabshots is relying on screenshots and image analysis to detect the changes made to a webpage. The reason is that in the websites that use dynamic refreshing of contents (Facebook, Linkedin, and Twitter to name a few), small changes could be considered major, leading to a large number of false positives.

NoTabNab [45] is a Firefox extension that records the favicon, title, and the positions of the topmost HTML elements of each webpage when the user navigates to the page for the first time. To detect tabnabbing attacks, NoTabNab computes the changes occurred in the layout of the page when the user navigates back to the inactive tab. However, it should be noted that the positioning of HTML elements of a webpage cannot be a determining factor for detecting tabnabbing attacks, because it changes if the page is resized. In addition, since only the topmost elements are considered, changes cannot be detected if all page contents are placed in an `<iframe>`.

An approach to perceive tabnabbing attack is proposed by Suri *et al.* [46]. The proposed approach leverages a signature-based technique for detecting tabnabbing attacks that are launched using iframes. For each webpage, the approach extracts the text content between script tags. The attack is then detected using a rule-based system which takes mouse clicks, `onblur` and `onfocus` events for iframes into account. The first limitation of this approach is that it is only designed for iframe-based tabnabbing attacks. Moreover, since only the contents of script tags are checked, the script-free variant of the attack cannot be detected using this approach. As for detecting the script-based variant of the attack, only inline scripts are considered, hence leaving out the external scripts undetected.

An anti-phishing framework is proposed by Sarika and Paul [47, 48] with the aim of defending against tabnabbing attacks. The approach first checks the URL of a webpage to determine whether it is blacklisted as a phishing website or not. It then continuously monitors the page using multi-agent systems to detect visual changes. The proposed multi-agent system consists of agents in three levels that can communicate with each other. If the degree of changes in a page is larger than a predefined threshold value, the page is recognized as tabnabbing and an alert message is displayed to the user.

3.4 Summary

In this chapter, we detailed the existing anti-phishing and anti-tabnabbing solutions. We first presented a categorization of phishing attacks and introduced some of the state-of-the-art techniques for phishing detection and prevention. We then described the existing anti-tabnabbing tools and techniques in two separate groups: script-blocking browser extensions and techniques specific to tabnabbing detection and prevention, and we called attention to the deficiencies of the solutions of each group. We believe that an effective tabnabbing detection and prevention technique should take more aspects of the tabnabbing attack into account. Moreover, it should actively prevent the attack and should not leave the decision of keeping the user's information safe into the hands of the user herself. We present such an approach in the next chapter (Chapter 4).

Chapter 4

Tabnabbing Detection Technique

In this chapter, we introduce **TabsGuard**, our hybrid tabnabbing detection and prevention approach. First, in Section 4.1, we present our problem statement with respect to the deficiencies of the existing anti-tabnabbing techniques as discussed in Chapter 3. In Section 4.2, we provide an overview of our approach. Then, in Section 4.3, we present the components of TabsGuard and provide details on TabsGuard's workflows for tabnabbing detection and prevention (Subsections 4.3.1 and 4.3.2).

4.1 Problem Statement

In the previous chapter (Chapter 3), we presented the existing anti-tabnabbing techniques and tools, and we pointed out their shortcomings in detecting tabnabbing attacks and actively preventing them. In particular, we discussed that most script-blocking browser extensions protect users against the script-based variant of the tabnabbing attack by disabling the execution of scripts on every website that is either blacklisted (in extensions that implement a blacklist-based approach) or not whitelisted (in extensions that implement a whitelist-based approach). This is not the most efficient and reliable approach as it requires the user to decide if she wants

JavaScript to run on a website or not. Furthermore, among the anti-tabnabbing tools of this group, only NoScript offers protection against the script-free variant of tabnabbing, which in our understanding, is not an effective prevention method as it merely postpones the launch of the attack in the hope that the user notices the attack and avoids entering her credentials into the website. We also reviewed four approaches and techniques specifically designed to detect and prevent tabnabbing attacks. Among those, NoTabNab [45], Suri *et al.*'s approach [46], and Sarika and Paul's approach [47, 48] report no qualitative or quantitative evaluation results indicating how good their approaches work in practice. TabShots [8], on the other hand, seems to be a promising anti-tabnabbing technique according to its evaluation results, but we find some drawbacks in TabShots (discussed in Section 3.3.2) that if addressed, more accurate detection results can be achieved.

In this thesis, we aim at designing a tabnabbing detection and prevention approach that is able to:

1. detect both script-based and script-free variants of the tabnabbing attack.
2. actively prevent the attack by navigating the user to a safe and trusted domain in case the attack is detected on the user's browser.
3. work on different categories of websites that the attack targets.
4. not impose a significant computational overhead on the browser.

Along the above line, in this chapter, we propose a tabnabbing detection and prevention approach, named TabsGuard. TabsGuard leverages a combination of heuristic-based metrics and data mining techniques to detect tabnabbing attacks. The following sections of this chapter present the details of our approach.

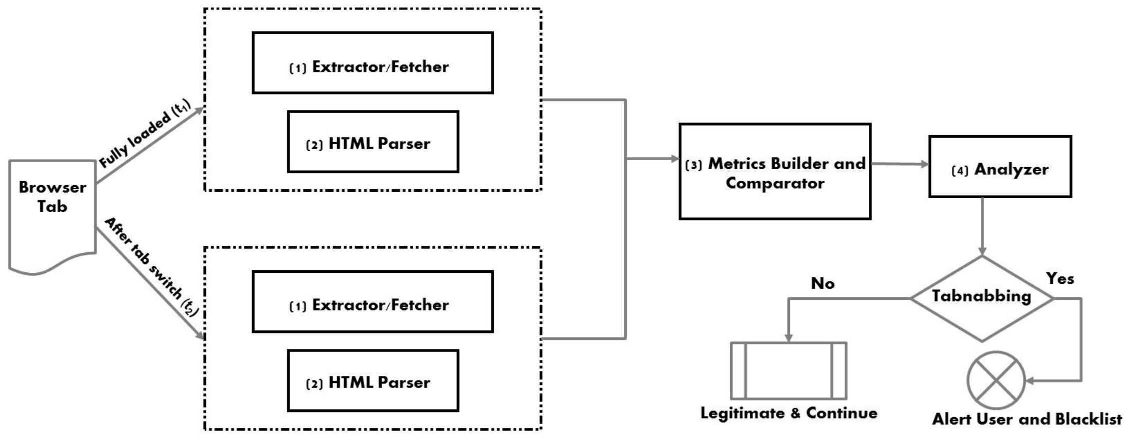


Figure 4.1: Overview of TabsGuard's workflow. The numbers indicate the order of execution.

4.2 Overview of TabsGuard

By definition, a tabnabbing attack changes the title, favicon, and contents of a page either by the use of scripts (*i.e.,* the script-based variant) or by refreshing the page after a predetermined interval (*i.e.,* the script-free variant). According to the tabnabbing attack scenario, changes made to the title, favicon, and contents of a webpage can best represent the changes made to the webpage under attack. Thus, an effective tabnabbing detection technique should be able to detect the degree of changes these three parameters go through when the tab is out of focus.

Figure 4.1 provides an overview of our proposed approach for tabnabbing detection and prevention. Our detection approach, given a page opened in a tab, fetches three parameters (*i.e.,* title, favicon, and the HTML Document Object Model (DOM) tree of the page) at two times t_1 and t_2 and compares the data for each parameter at time t_1 with its corresponding parameter at time t_2; where t_1 is the time when the page is fully loaded and t_2 is the time when the user returns to that tab after at least one tab switch. This comparison is used for monitoring the changes that the page goes through between times t_1 and t_2. The comparison results are then analyzed using data mining techniques, and the page is eventually detected as tabnabbing or legitimate. Our prevention approach, given a page detected as tabnabbing, displays an alert message to the user and to prevent the user from further proceeding into the tabnabbing website, redirects the user to a trusted domain and adds the page to a local blacklist in the user's browser.

4.3 Components of TabsGuard

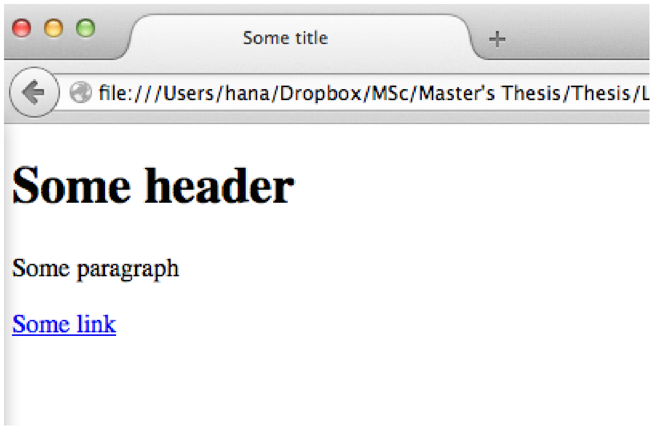
In this section, we detail the components that our tabnabbing detection and prevention approach consists of. As shown in Figure 4.1, the design and implementation of TabsGuard consists of the following phases: (1) Fetching, (2) Parsing, (3) Metrics building and comparison, and (4) Analysis. Subsections 4.3.1 and 4.3.2 describe TabsGuard's tabnabbing detection and prevention workflows, respectively.

4.3.1 TabsGuard's Detection Workflow

Let us assume we have a webpage open in a browser tab. In phase 1, the extractor fetches the title and favicon of the page. In phase 2, the parsing engine takes the HTML source code of the page, extracts pure HTML from the tags, and removes the text contents and CSS styles to get the DOM tree representation we require for the next phase.

The DOM is a programming interface for HTML that provides a structured representation of the document as a tree of nodes and objects with properties, methods, and events. With the DOM, all nodes in an HTML document can be accessed or manipulated using JavaScript [51, 52, 53]. Figure 4.2 and Listing 4.1 indicate a simple HTML page and its source code, respectively. The HTML DOM tree of this webpage is shown in Figure 4.3.

It should be noted that DOM trees have traditionally been used to model the structural information of web documents. In particular, in the approach proposed by Rosiello *et al.* [54], DOM-based layout comparisons are made between legitimate websites and potential phishing websites to detect phishing attacks. In our tabnabbing detection approach, similarly, we use the HTML DOM tree to model the HTML layout of a webpage. We believe that the major changes made to the HTML DOM tree of the webpage during the time the page is out of focus would modify the overall structure of the page and make the page a tabnabbing candidate.



Some header

Some paragraph

Some link

Figure 4.2: A simple HTML page.

Listing 4.1 Source code of the HTML page in Figure 4.2

```
<html>
    <head>
        <title>Some title</title>
    </head>
    <body>
        <h1>Some header</h1>
        <div>
            <p>Some paragraph</p>
        </div>
        <a href="Some URL">Some link</a>
    </body>
</html>
```

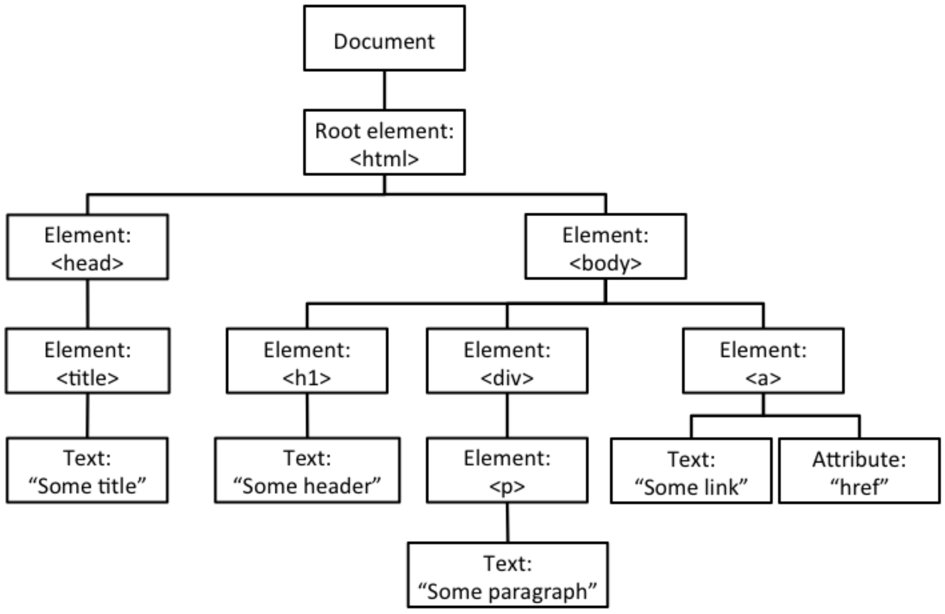


Figure 4.3: HTML DOM tree of the webpage in Figure 4.2.

In phase 3, using the HTML content of the page, the metrics builder computes a number of heuristic-based metrics for comparing the DOM trees built at times t_1 and t_2. For this purpose, we studied a number of existing works [55, 56] used to detect layout/structural similarity between webpages and text documents. As presented by Ricca and Tonella [55, 56], there are three different ways to classify similar pages.

1. *Textual identity*: Two pages are considered identical if they have the *exact same* HTML source code.
2. *Syntactical identity*: Syntax trees of the pages are compared, text content between HTML tags is removed, and pages with the *exact same* structure are considered identical.
3. *Syntactical similarity*: A similarity metric is computed based on the syntax trees of the pages, and pages with *similar* structure are classified.

The limitation of the first two classification approaches is that they look for exact matches. Syntactical similarity, on the other hand, can be used to define a similarity threshold according to which two pages are considered similar. In our comparison, we do not expect the webpage at time t_1 to have the exact same HTML source code or structure as the one at time t_2, but that the two webpages have similar structures. Thus, we define the following heuristic-based metrics for conducting the comparison with respect to syntactical similarity.

H1) Common Paths (CP) [57]: We use CP for comparing the structure of two webpages by looking at the paths leading from the root node to the leaf nodes in the HTML DOM tree. A path consists of the concatenation of the names of the elements it passes from root to leaf. Each page can then be represented in terms of the set of paths it contains (*i.e.,* $p(P)$). Equation 4.1 illustrates the CP similarity measure, where P_1 is the page at time t_1 and P_2 is the page at time t_2. The closer a CP value to 1, the higher the degree of similarity between P_1 and P_2.

$$CP(P_1, P_2) = \frac{|p(P_1) \cap p(P_2)|}{max(|p(P_1)|, |p(P_2)|)} \tag{4.1}$$

H2) Cosine Similarity (CS) [58] and *Class names:* The cosine measure similarity is a similarity metric that has been used for detecting the similarity between text documents. We derive a vector of tags from the page at both times t_1 and t_2. For each tag that uses a class name, we append the class name to its corresponding tag name as well. Therefore, the page at times t_1 and t_2 is viewed as two vectors in a vector space, with each {tag-class} name having its own axis. Let P_1 be the page at time t_1 and P_2 be the page at time t_2. Cosine similarity is computed by dividing the dot product of P_1 and P_2 (the numerator in Equation 4.2) by the product of their magnitudes (the denominator in Equation 4.2). A CS value closer to 1 indicates more similarity between the two pages.

$$CS(P_1, P_2) = \frac{P_1.P_2}{|P_1||P_2|} = \frac{\sum_{i=1}^{n} P_{1_i} \times P_{2_i}}{\sqrt{\sum_{i=1}^{n}(P_{1_i})^2} \times \sqrt{\sum_{i=1}^{n}(P_{2_i})^2}} \tag{4.2}$$

H3) Tag Frequency Distribution Analysis (TFDA) [59, 60]: TFDA is proposed with the assumption that tag frequencies reflect built-in characteristics of a webpage and are closely related to its structure. To calculate this metric, we compute the frequency of each HTML tag and incorporate them into Equation 4.3, where F_{t_1} and F_{t_2} are the frequencies of tag t in pages P_1 and P_2, n is the total number of tags, w_t is the weight for the t^{th} tag and $\sum_{t=1}^{n} w_t = 1$. TFDA values are then normalized to fall between 0 and 1. A TFDA value closer to 1 shows that P_2 resembles P_1 to a great extent.

$$TFDA(P_1, P_2) = \sum_{i=1}^{n} (F_{t_1} - F_{t_2})^2 \times w_t \tag{4.3}$$

H4) Input fields added to the page: In most cases, a malicious page can be a threat to a user only if it requires certain actions from the user (*e.g.,* user input). In particular, the page that appears in the inactive tab after tab switch most of the time contains input fields that ask for user's personal or financial information. These input fields could belong to an email service login form, a bank website login form, a sign up form, etc. Therefore, when comparing P_1 and P_2, we take the new `<input>` elements added to the page at time t_2 into account. We indicate this using true/false, where true means that P_2 has additional `<input>` elements compared to P_1, and false means that P_1 and P_2 have an equal number of `<input>` elements.

H5) iframes in the page before and after tab switch: The tabnabbing attack can also be launched using iframes. Hence, if a page consists of `<iframe>` elements at both times t_1 and t_2, we consider it a tabnabbing candidate. We indicate this using true/false, where true means that both P_1 and P_2 contain `<iframe>` elements, and false means otherwise.

The abovementioned metrics provide values for comparing the HTML DOM trees of the page at two times t_1 and t_2. *H1*, *H2*, and *H3* report the comparison results in terms of numerical values, whereas *H4* and *H5* provide truth values. This comparison helps us detect any changes made to the HTML layout or the overall structure of the page during the time it was out of focus. The comparator then determines the percentage of similarity between the titles at times t_1 and t_2 using the algorithm of Longest Common Substring (LCS).

The LCS problem is the problem of finding the longest string that is a substring of two or more strings [61]. This problem can be solved by finding the longest common suffix for all pairs of prefixes of the strings using dynamic programming. Dynamic programming is an algorithmic technique for solving complex problems by breaking them down into smaller subproblems and constructing a subsolution of the problem from previously found subsolutions [62].

The reason we use LCS for comparing the titles at times t_1 and t_2 is that some trusted websites use dynamic refreshing of contents and automatically update their title with respect to the dynamic changes. For instance, the title of a Facebook page — `Facebook` — might change to `Facebook(1)` between times t_1 and t_2 due to a new notification or message. In this case, the LCS algorithm finds `Facebook` with the length of 8 as the longest common substring between the two titles. To find the percentage of similarity between the two titles, this number is divided by the length of the longer title (11) and multiplied by 100, resulting in an approximate similarity of 73%. In general, given the two titles T_1 and T_2, we use Equation 4.4 to calculate the percentage of similarity between the titles at times t_1 and t_2.

$$Similarity(T_1, T_2) = \frac{LCS(T_1, T_2)}{max(T_1, T_2)} \times 100 \tag{4.4}$$

As the final step of phase 3, the comparator compares the URL of each favicon at times t_1 and t_2 to look for the changes made to the favicon after tab switch. Since in most cases two favicons cannot have the same source, we believe URL comparison is sufficient for realizing whether the favicon has changed after tab switch or not. We indicate favicon similarity using true/false values, where true means the favicons at two times t_1 and t_2 are the same, and false means otherwise.

Once the comparison is completed, we need to analyze the comparison results to determine whether the changes occurred to a webpage are due to a tabnabbing attack or not. We do this using anomaly detection techniques. Anomaly detection is the identification of cases or items that are unusual within seemingly homogeneous data and/or do not conform to an expected behavior [63, 64]. An anomalous item, also known as an outlier, is an item that is considered unusual due to its difference from the normal distribution of data. To find the outliers in a given dataset, a nearest neighbor-based model can be used to score the input data. Such model detects the outliers in the dataset using the adjacent items that define a neighborhood. In our case, outliers represent tabnabbing pages, and our aim is to find them in a given dataset. Hence, for each page in the dataset, if the outlier score determined with respect to the degree of changes is higher than the average outlier score, the page is detected as tabnabbing. Otherwise, it is detected as legitimate. The results obtained from using anomaly detection techniques can then help us define threshold values for each of the heuristic-based metrics. Using these threshold values, we can detect whether a webpage is tabnabbing or legitimate. Among the anomaly detection techniques proposed in the literature, we have selected the following:

1. *k-NN*: One of the techniques that has been widely used for anomaly detection is the k-Nearest Neighbors (k-NN) algorithm, using which one can measure whether or not an item is distant from the other items in a dataset. According to this algorithm, the outlier score of each item in the dataset is determined by the distance to its k-nearest neighbor [15].

2. *LOF*: The Local Outlier Factor (LOF) is a local density-based algorithm, where local density is estimated using the distance at which an item can be reached from its k-nearest neighbors. LOF is based on the concept of local outliers, that

is assigning each item in a dataset a factor of being an outlier depending on their neighborhood. Using LOF, one can find the outliers in a dataset by measuring the local deviation of an item with respect to its neighbors. This algorithm considers an item to be normal if it has an outlier score of approximately 1 and considers it an outlier if it has an outlier score greater than 1 [16, 65].

3. *COF*: The Connectivity-Based Outlier Factor (COF) is a modification of the LOF algorithm proposed with the purpose of handling outliers deviating from low density patterns. Same as LOF, COF considers an item to be normal if it has an outlier score of approximately 1 and considers it an outlier if it has an outlier score greater than 1 [17].

4. *INFLO*: The Influenced Outlierness (INFLO) is a modification of the LOF algorithm that measures local outliers based on a symmetric neighborhood relationship. This is achieved by expanding the neighborhood set and considering both neighbors and reverse neighbors when estimating the local density of a given item in a dataset [18, 65].

5. *LOCI*: The Local Correlation Integral (LOCI) is a local density-based algorithm that improves the previous algorithms in different aspects. Most significantly, it automatically flags outliers by providing a cut-off point based on probabilistic reasoning. The proposed cut-off point is 3 [66].

6. *LoOP*: The Local Outlier Probability (LoOP) is a local density-based algorithm that represents the outlier score as the probability of an item being an outlier. The outlier score is thus independent from the distribution of the data. In addition, the score being provided as a probability makes it more interpretable

and facilitates the comparison of score values over multiple datasets [67].

7. *HBOS*: The Histogram-Based Outlier Score (HBOS) is a statistical-based algorithm that calculates the outlier score by creating a histogram. HBOS assumes independence between features, models each feature by a separate histogram, and aggregates the results to calculate the outlier score [18, 68].

All the abovementioned techniques are nearest neighbor-based algorithms except for HBOS. Nearest neighbor-based algorithms are widely used for unsupervised anomaly detection, which is an unsupervised learning approach that detects outliers in a dataset without having any information about the dataset beforehand, as opposed to supervised anomaly detection that requires a training dataset to learn a model from. Unsupervised anomaly detection techniques do not need prior training. Such techniques are used when only a small fraction of the items in the dataset are anomalous and these anomalous items are significantly different from the normal items in the dataset. Supervised anomaly detection techniques, on the other hand, are used when a balanced dataset containing roughly the same number of normal and anomalous items is available.

We have chosen unsupervised anomaly detection techniques over supervised anomaly detection techniques, because the number of anomalous items (*i.e.,* tabnabbing pages) available to us is very small compared to the number of normal items (*i.e.,* legitimate pages) that we can access. As explained earlier, there are only two proof of concepts available for the tabnabbing attack [7, 9]. To address the shortage of tabnabbing pages, we generate more tabnabbing pages for our evaluation purposes so that we can have a dataset consisting of 1% tabnabbing pages.

4.3.2 TabsGuard's Prevention Workflow

After the tabnabbing detection process is completed, we begin the prevention process with respect to the detection results. If a page is detected as legitimate, we take no further steps in order not to interfere with the user's normal browsing and the user will be free to proceed or input her information into the website. If the page is detected as tabnabbing, we first warn the user by displaying an alert message. We then ask the user to add the page to a blacklist, which in turn adds the page to a local blacklist in the user's browser so that the user will be warned and prevented from proceeding further the next time she tries to access the tabnabbing page. After having the tabnabbing page blacklisted, we redirect the user to a trusted domain.

Our prevention approach is active as opposed to most phishing (and tabnabbing) prevention approaches that are passive, that is to say, when we identify the threat, we do not rely on the user's decision to proceed to the malicious website or not. Instead, we block access to the website and actively prevent the attack. More specifically, for each webpage that is detected as a potential tabnabbing attack, we first display an alert message to the user and ask her to blacklist the page. This is a precautionary measure as blacklisting the page stores its information in a database on the user's browser so that further access is denied to this webpage. Once the blacklisting of the page is done, the user is redirected to a safe page such as Google, Yahoo, or a website of the user's choice.

4.4 Summary

In this chapter, we presented our tabnabbing detection and prevention approach, named TabsGuard. TabsGuard is a hybrid anti-tabnabbing approach as it combines heuristic-based metrics and anomaly detection techniques. We first described our problem statement by calling attention to the lack of an effective and efficient tabnabbing detection and prevention approach. We presented an overview of TabsGuard and then introduced TabsGuard's components and separately described TabsGuard's detection and prevention workflows in extent. We presented the heuristic-based metrics that we use for determining the degree of changes made to a tabnabbed page during the time it is out of focus. We then detailed the anomaly detection techniques that we use for analyzing these changes and determining the final detection result that whether a page is tabnabbing or not. Finally, we described TabsGuard's prevention workflow and explained why it is superior to the other prevention approaches proposed so far. In the next chapter (Chapter 5), we will present the details of TabsGuard's evaluation process and results. We will also analyze our results and compare TabsGuard's effectiveness in defending against tabnabbing attacks with the existing techniques.

Chapter 5

Experimental Evaluation

In the previous chapter, we presented our anti-tabnabbing approach and tool (named TabsGuard) and detailed TabsGuard's detection and prevention mechanism. In this chapter, first we provide an overview of the way TabsGuard is evaluated (Section 5.1). Second, in Section 5.2, we discuss the experimental setup along with the toolsets used. Section 5.3 explains the experiments we conduct for evaluating TabsGuard's capability in detecting tabnabbing attacks along with TabsGuard's performance. In this section, we also report our experimental results. We conduct a comparative analysis with the existing techniques in Section 5.4. Finally, in Section 5.5, we analyze the impact of each of the five heuristic-based metrics we proposed in Chapter 4 on a page being legitimate or tabnabbing.

5.1 Implementation and Objectives

We have implemented TabsGuard as a Mozilla Firefox (Version 29.0.1) extension. To this end, we first implemented a prototype of our tabnabbing detection and prevention approach using JavaScript. In this implementation, we first capture the title, favicon, and the HTML DOM tree of a given webpage in the browser when the `onload` event [69] of a page is triggered (*i.e.,* when the page is fully loaded). We then use `Window.onblur` [70] and `Window.onfocus` [71] properties to detect when a tab switch occurs. The `onblur` event is triggered when the current window loses focus, and the `onfocus` event is triggered when the user sets focus on the current window. Upon the firing of the `onfocus` event, we capture the title, favicon, and the HTML DOM tree of the page once more and compare this data with the corresponding data captured at the time of the `onload` event. The comparison is carried out using the five heuristic-based metrics and the techniques for comparing titles (*i.e.,* LCS) and favicons (*i.e.,* URL comparison) as discussed in Chapter 4.

In order to extend this functionality to all the open tabs in a browser, we deployed our implementation on Mozilla Firefox as an extension with respect to the Mozilla extension development guidelines [72]. To allow for tracking tab switch events on all the open tabs, we use *Progress Listeners* [73] instead of `onblur` and `onfocus` events. In our case, progress listeners enable the extension to get notified of the loading of a page in the browser and tab switch events. The extension records the title, favicon, and the HTML DOM tree of each page open in a tab once the page is fully loaded and once when the user returns to the tab after having moved away from the tab earlier (*i.e.,* after a tab switch event is detected on that tab). In case the tabnabbing attack is detected on a tab, we use a *notification box* [74] to alert the user and ask her to blacklist the website.

Our evaluation of TabsGuard consists of three parts. First, we evaluate the capability of TabsGuard in detecting tabnabbing attacks. Second, we evaluate TabsGuard's effectiveness for detecting diverse tabnabbing attacks. Third, we evaluate

TabsGuard from the performance point of view. For the first part, we focus on addressing two points:

a. Using TabsGuard for correctly identifying tabnabbing pages as malicious and measuring the true positive rate;

b. Using TabsGuard for correctly identifying legitimate pages as benign and measuring the false positive rate.

To this end, we use the top 1,000 legitimate pages from Alexa [14] and 11 tabnabbing pages generated by ourselves [75]. The combined dataset of legitimate and tabnabbing pages is the basis for the evaluation of our approach and tool when using each one of the seven anomaly detection techniques presented in Section 4.3.1.

To evaluate TabsGuard's effectiveness for detecting tabnabbing attacks that target trusted websites of different categories, we use the 11 tabnabbing pages generated by ourselves. In particular, we use the most widely adopted open-source web applications based on which millions of customized web applications are deployed on the Web. To balance the mix of application types, we consider three distinct classes of applications (*i.e.,* email services, social networking websites, and popular everyday-use websites) and targeted the tabnabbing attack on them.

To evaluate TabsGuard's performance as a Firefox extension, we focus on evaluating the overhead TabsGuard imposes on the browser by measuring its impact on the browser startup and execution time; The former is measured using the *About Startup* Firefox extension [76] and analyzed using the *Wilcoxon Rank-Sum Test* [77], and the latter is measured using the amount of time it takes for certain websites to load. We conduct the performance experiments a) having TabsGuard installed on the browser, b) having the three script-blocking Firefox extensions (NoScript, Controle de Scripts, and YesScript) installed on the browser, and c) having a browser with no extensions installed. We then measure the overhead TabsGuard and the other extensions impose on the browser and compare the results.

5.2 Experimental Setup

The toolsets we use for evaluating TabsGuard include: RapidMiner 5.3 [50] for facilitating the data mining process, iMacros [78] for the automation of repetitious tasks on the browser during the data collection step, and Firebug [79] for our performance tests. RapidMiner is a software platform that provides an environment for predictive analytics, data mining, machine learning, and etc. iMacros is an extension developed for Firefox, Chrome, and Internet Explorer, providing the ability of recording and replaying repetitious tasks on the browser. Firebug is a web development tool that enables easy monitoring and debugging of websites.

The data we use consists of top 1,000 Alexa websites and 11 tabnabbing pages generated by ourselves. To build a complete dataset, we run our extension on these 1,011 websites to get the computed values for the five heuristic-based metrics, the percentage of title similarity, and the similarity of favicons (indicated by true or false). We develop an in-house script (Listing 5.1) to automatically create the dataset which is then fed to iMacros. In order to prevent from slowing down or crashing the browser, instead of once opening 1,000 tabs in the browser, we split the 1,000 legitimate websites into ten lists. Each list is given to the script as input in one separate run. The 11 tabnabbing pages are also given to the script as an individual list in a separate run.

Listing 5.1 Script for automating data collection

```
//Phase 1
1. For each website in the list
2.    Open the website in a new tab
3.    Wait until the page is fully loaded
//Phase 2
4. Open a new tab
5. For each website open in a tab
6.    Go to the website
7.    Go to the blank tab
8.    Wait for 60 seconds
9.    Go back to the website
```

When our in-house script is executed, first, each website is opened in a tab, and after being fully loaded, its title, favicon, and HTML layout are recorded. After this step, a blank tab is opened to serve as a temporary location the browser is redirected to at the time of tab switches. Then, for every website, the browser is redirected to its corresponding tab, switches location to the blank tab, and after staying on the blank tab for 60 seconds, is again redirected to the former tab. At this point, the title, favicon, and HTML layout of the page are recorded again, and the heuristic-based metrics are computed. Then, the script moves on to the next page and so on. When this procedure is done for all the open tabs, the script execution is completed. Finally, a dataset consisting of the computed values of the five metrics, title, and favicon for all 1,011 websites is created. The complete dataset is then passed to an anomaly detection algorithm as input to determine the outlier score using which the false positives are identified.

As noted above, we set 60 seconds as the amount of waiting time required before switching back to each tab. The reason is that from the attacker's point of view, each page is reloaded with new contents either when the tab switch event is detected (in the script-based variant of the attack) or after a predetermined time interval (in the script-free variant of the attack). In the first case, reloading of the contents is done shortly after the user moves away from the tab. In the second case, page refresh should be done no later than 60 seconds after the user has switched tabs, because according to Nielsen [49], *"The average page visit lasts a little less than a minute."* Therefore, assuming that the user visits only one other page and then decides to return to the inactive tab, the interval is less than 60 seconds, meaning that the attacker has 60 seconds at most to reload the tab with a new appearance. Hence, in our detection scenario, we assume that the user returns to the tab after 60 seconds, and we measure the changes after the 60-second interval has elapsed.

5.3 Experiments

In this section, we describe the three sets of experiments we conduct for evaluating our anti-tabnabbing approach and report their results. First, we describe the set of experiments that evaluate the effectiveness of TabsGuard's tabnabbing detection mechanism. Here, we use our complete dataset consisting of the computed values for the heuristic-based metrics, title, and favicon of our test pages on several anomaly detection techniques. Our goal is to obtain the outlier score for the items in our dataset for each anomaly detection technique and then compare the results for all techniques with each other. In this way, we can determine which anomaly detection technique(s) is(are) the most accurate ones to be included in the tabnabbing detection process. Second, we evaluate the capability of TabsGuard in detecting different categories of tabnabbing websites. Third, we describe the set of experiments we use for evaluating TabsGuard's overhead. Our objective in this set of experiments is to measure the overhead TabsGuard, as a Firefox extension, has on the browser and compare it with the overheads measured for the script-blocking browser extensions that detect tabnabbing.

5.3.1 Effectiveness of the Detection Mechanism

Here, we describe our experiments for evaluating and comparing different anomaly detection techniques. The results obtained shed light on TabsGuard's capability of detecting tabnabbing attacks when using each anomaly detection technique. To this end, we import our complete dataset to RapidMiner and use RapidMiner operators for k-NN, LOF, COF, INFLO, LOCI, LoOP, and HBOS, respectively to measure the ourlier score for the 1,011 webpages in our dataset. We then report the results we obtain using each anomaly detection technique.

The results are reported in terms of the average outlier score determined by the process, the number of outliers (*i.e.,* the number of pages identified as tabnabbing), False Positive Rate or FPR (Equation 5.1), detection rate or True Positive Rate or TPR (Equation 5.2), and $Accuracy$ (Equation 5.3). In these equations, FP is the number of false positives (*i.e.,* the number of legitimate pages that are incorrectly identified as tabnabbing), TN is the number of true negatives (*i.e.,* the number of legitimate pages that are correctly identified as benign, TP is the number of true positives (*i.e.,* the number of tabnabbing pages correctly identified as tabnabbing), and FN is the number of false negatives (*i.e.,* the number of tabnabbing pages incorrectly identified as benign). The five heuristic-based metrics introduced in Section 4.3.1 as well as the title and favicon of each data item serve as our attributes for this set of experiments. We also define a label attribute, named “Tabnabbing” which is set to false for all Alexa websites and to true for the 11 tabnabbing pages in the dataset.

It is noteworthy that FPR and TPR are good indicators of the performance of a system and are widely used in reporting the results of phishing (and tabnabbing) detection techniques. Together, they measure the percentage of attacks a detection technique can correctly detect in addition to the percentage of the incorrectly detected legitimate websites. There exists a trade-off between these two indicators, in that an attempt to increase TPR may result in an increase in FPR, and an attempt to decrease FPR may lead to a lower value for TPR [80].

$$FPR = \frac{FP}{FP + TN} \tag{5.1}$$

$$TPR = \frac{TP}{TP + FN} \tag{5.2}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.3}$$

We consider a webpage in our dataset an outlier if the outlier score computed for it is greater than the average outlier score (*i.e.,* $outlierscore > average$). One can also consider a webpage in our dataset an outlier if the outlier score computed for it is greater than the average outlier score *plus* the variance or if the outlier score computed for it is less than the average outlier score *minus* the variance (*i.e.,* $outlierscore > average + variance$ or $outlierscore < average - variance$). Figure 5.1 illustrates these two definitions for a sample chart. Using the first definition, outliers are the items in the area annotated by 1, whereas using the second definition, outliers are the

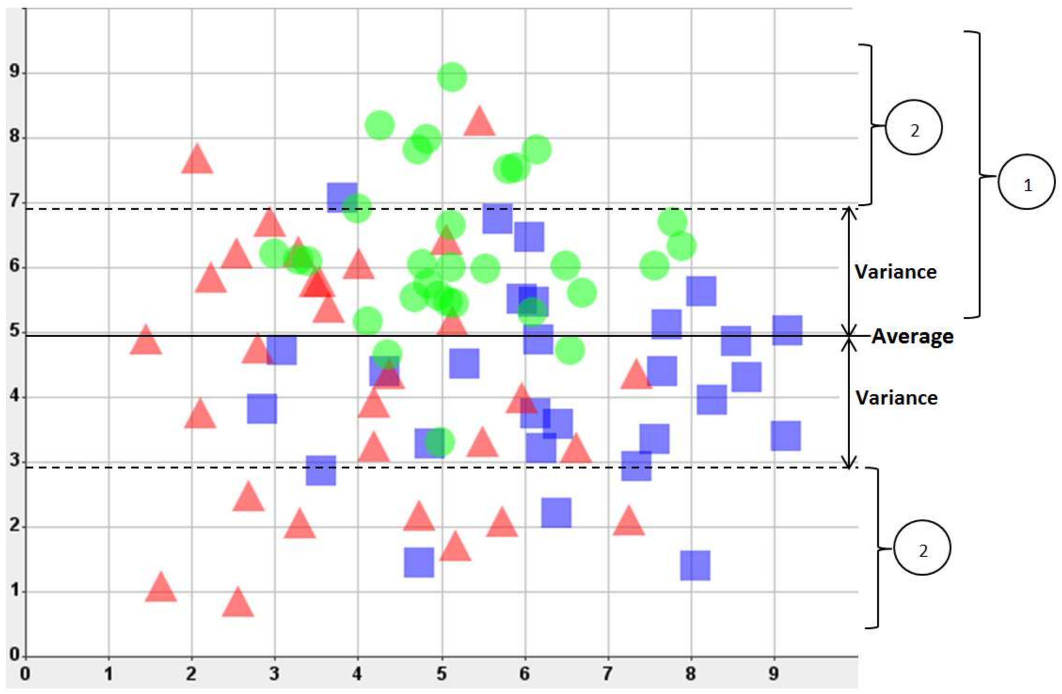


Figure 5.1: Two definitions for outliers in an example

items in the area annotated by 2. As one might notice, using the second definition, fewer outliers are identified, especially in the cases where $average - variance$ (*i.e.,* $average\ minus\ variance$) is a number out of the scope of the outlier scores computed for the dataset — considering the fact that the outlier score for an item is always a non-negative number. We determine the number of outliers (*i.e.,* the number of pages in our dataset identified as tabnabbing) for each anomaly detection technique using both definitions. However, to consider the worst case, we report the number of correctly identified tabnabbing pages (*i.e.,* true positives) and compute FPR, TPR, and $Accuracy$ with respect to the first definition. It should be noted that we report the resulted values for FPR, TPR, and $Accuracy$ as percentages.

Table 5.1: Outlier information provided by the seven anomaly detection techniques.

	Average Outlier Score and Variance	**Number of Outliers by Definition 1**	**Number of Outliers by Definition 2**
k-NN	0.008 ± 0.064	28	17
LOF	8933.269 ± 80692.960	13	12
COF	1.679 ± 11.702	38	17
INFLO	8920.857 ± 80749.120	13	12
LOCI	3	14[1]	-
LoOP	0.014 ± 0.109	52	16
HBOS	0.131 ± 0.890	27	27

Results

Table 5.1 shows the average outlier score and variance of each of the seven anomaly detection techniques as well as the number of outliers determined by each technique with respect to both definitions of outliers. The number of outliers determined by each anomaly detection technique represents the number of webpages in our dataset that the anomaly detection technique identifies as tabnabbing pages.

As described in Chapter 4, we use k-NN to measure whether an item is distant from the other items in a dataset. The outlier score of each item in the dataset is determined by the distance to its k-nearest neighbor. In RapidMiner, k-NN Global Anomaly Score serves as an operator for k-NN anomaly detection technique that calculates the outlier score. Here, the outlier score is the average of the distance to the nearest neighbors, and the higher the score the more anomalous the instance is. Parameter k defines the number of neighbors to be considered, which is set to 10 in our experiment. This is the number we find optimized by manually checking some random

[1]This number is determined by the proposed cut-off point which is 3, but shown in the column for definition 1.

numbers of k between 2 and 15. Using the k-NN Global Anomaly Score operator, the average outlier score obtained is 0.008 with the variance of 0.064, denoted as 0.008 ± 0.064. This results in 28 outliers considering the first definition of outliers, and 17 outliers considering the second definition of outliers. All 11 tabnabbing pages are correctly identified as outliers by this operator.

Using the Local Outlier Factor (LOF) operator, the average outlier score obtained is 8933.269 with the variance of 80692.960, denoted as 8933.269 ± 80692.960. This results in 13 outliers considering the first definition of outliers, and 12 outliers considering the second definition of outliers. None of the 11 tabnabbing pages are identified as outliers.

Using the Connectivity-Based Outlier Factor (COF) operator, the average outlier score obtained is 1.679 with the variance of 11.702, denoted as 1.679 ± 11.702. This results in 38 outliers considering the first definition of outliers, and 17 outliers considering the second definition of outliers. This algorithm correctly identifies only 2 of the 11 tabnabbing pages.

Using the Influenced Outlierness (INFLO) operator, the average outlier score obtained is 8920.857 with the variance of 80749.120, denoted as 8920.857 ± 80749.120. This results in 13 outliers considering the first definition of outliers, and 12 outliers considering the second definition of outliers. None of the 11 tabnabbing pages are identified as outliers.

Using the Local Correlation Integral (LOCI) operator, 14 outliers are identified with respect to the cut-off point (*i.e.,* 3) proposed by the algorithm. None of the 11 tabnabbing pages are identified as outliers; however, all the tabnabbing pages have outlier scores very close to 3, ranging from $[2.964, 2.988]$.

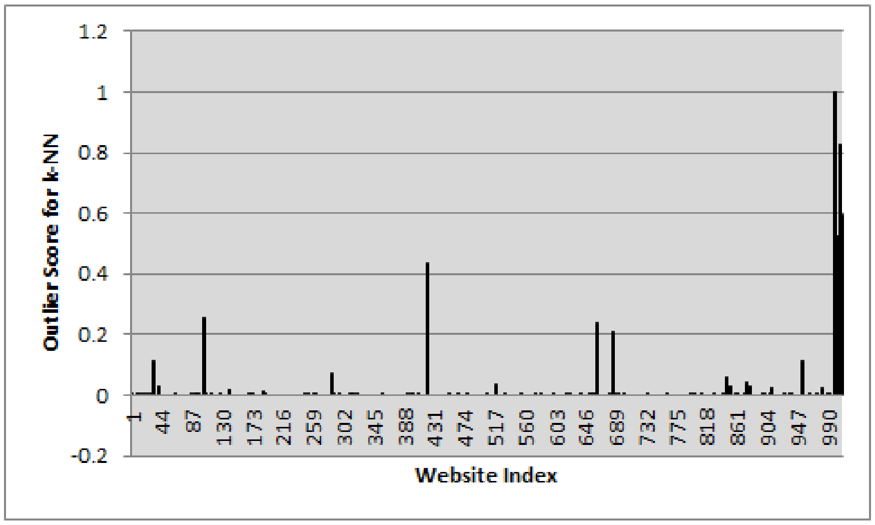


Figure 5.2: Outlierness of legitimate and tabnabbing websites measured by k-NN.

Using the Local Outlier Probability (LoOP) operator, the average outlier score obtained is 0.014 with the variance of 0.109, denoted as 0.014$\pm$0.109. This results in 52 outliers considering the first definition of outliers, and 16 outliers considering the second definition of outliers. This algorithm correctly identifies only 2 of the 11 tabnabbing pages.

Using the Histogram-based Outlier Score (HBOS) operator, the average outlier score obtained is 0.014 with the variance of 0.109, denoted as 0.131$\pm$0.890. This results in 27 outliers considering the first and second definitions of outliers. All 11 tabnabbing pages are correctly identified as outliers by this operator.

To clarify the results obtained and to provide a means for comparing them, we introduce a concept, named *outlierness*. Outlierness is defined as the degree of being an outlier, which is indicated by the outlier score for each item in a dataset [81]. This concept is widely used for illustrating the distribution of outliers in a dataset. The

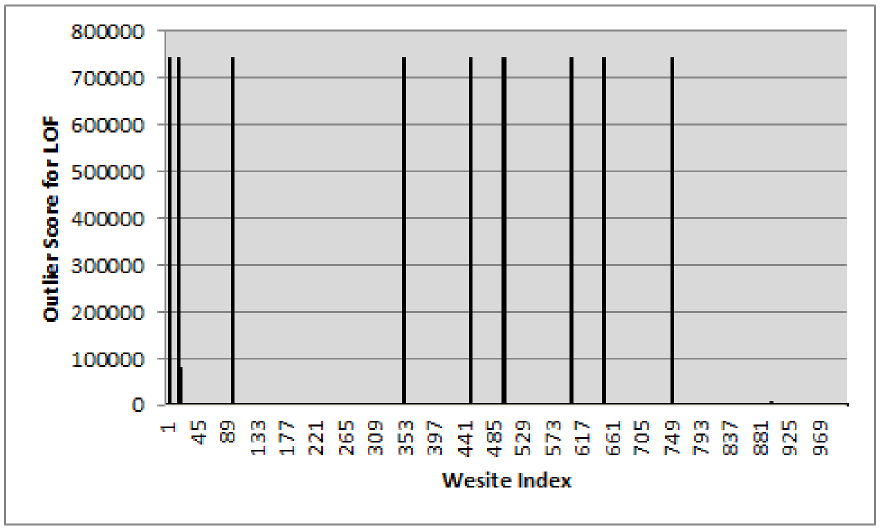


Figure 5.3: Outlierness of legitimate and tabnabbing websites measured by LOF.

following charts (Figures 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, and 5.8) demonstrate the outlierness of the websites in our dataset when using the seven abovementioned anomaly detection techniques. In our dataset, the 1,000 legitimate websites from Alexa come first, and the 11 tabnabbing pages are placed in indexes $[1001, 1011]$ for ease of presentation. It should be noted that placing the 11 tabnabbing pages at the end of the dataset does not affect the results determined by the anomaly detection techniques, because anomaly detection techniques do not consider the order of data when computing the outlier score for each webpage in the dataset.

To be able to compare each anomaly detection technique's capability of correctly identifying the tabnabbing pages in the dataset, we compute FPR (Equation 5.1), TPR (Equation 5.2), and $Accuracy$ (Equation 5.3) for each technique. Table 5.2 summarizes the resulted values for each anomaly detection technique. Figures 5.9, 5.10, and 5.11 demonstrate these values for the seven anomaly detection techniques.

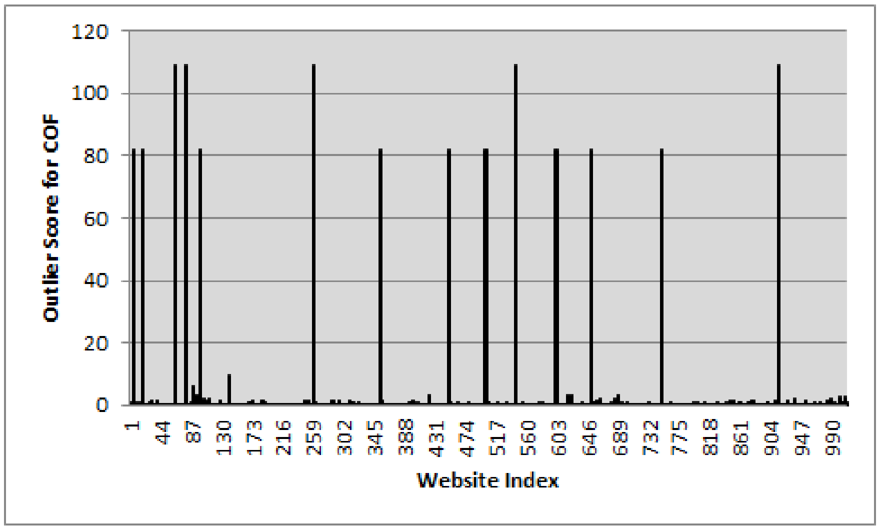


Figure 5.4: Outlierness of legitimate and tabnabbing websites measured by COF.

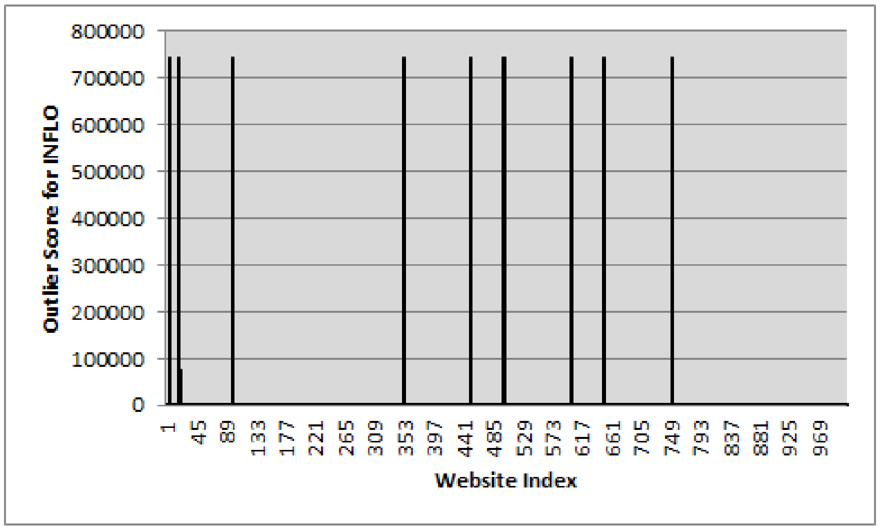


Figure 5.5: Outlierness of legitimate and tabnabbing websites measured by INFLO.

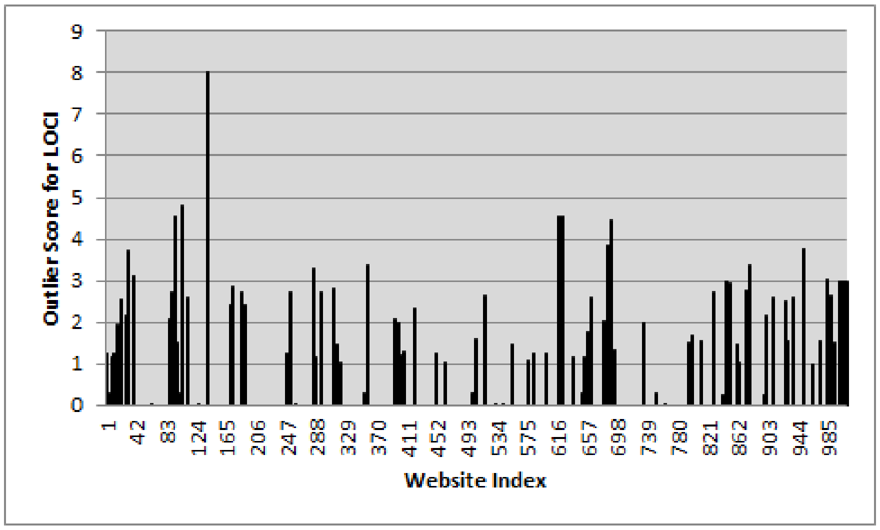


Figure 5.6: Outlierness of legitimate and tabnabbing websites measured by LOCI.

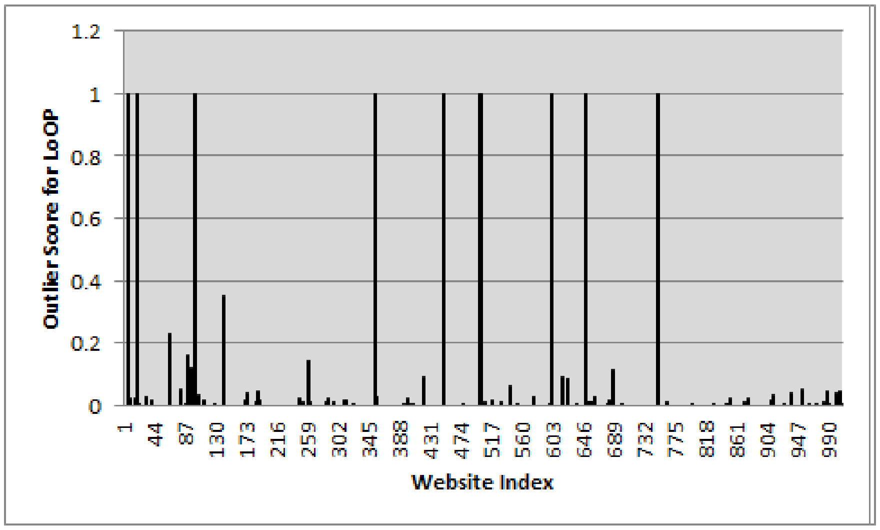


Figure 5.7: Outlierness of legitimate and tabnabbing websites measured by LoOP.

What we learn from these figures is that local density-based outlier detection techniques (*i.e.,* LOF, COF, INFLO, LOCI, and LoOP) do not work well in our case, where the problem is finding the tabnabbing pages in the whole dataset. Local

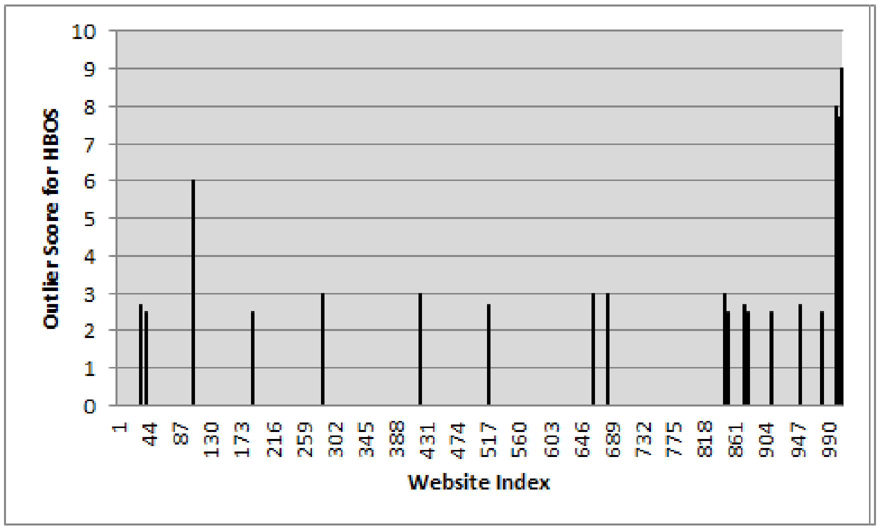


Figure 5.8: Outlierness of legitimate and tabnabbing websites measured by HBOS.

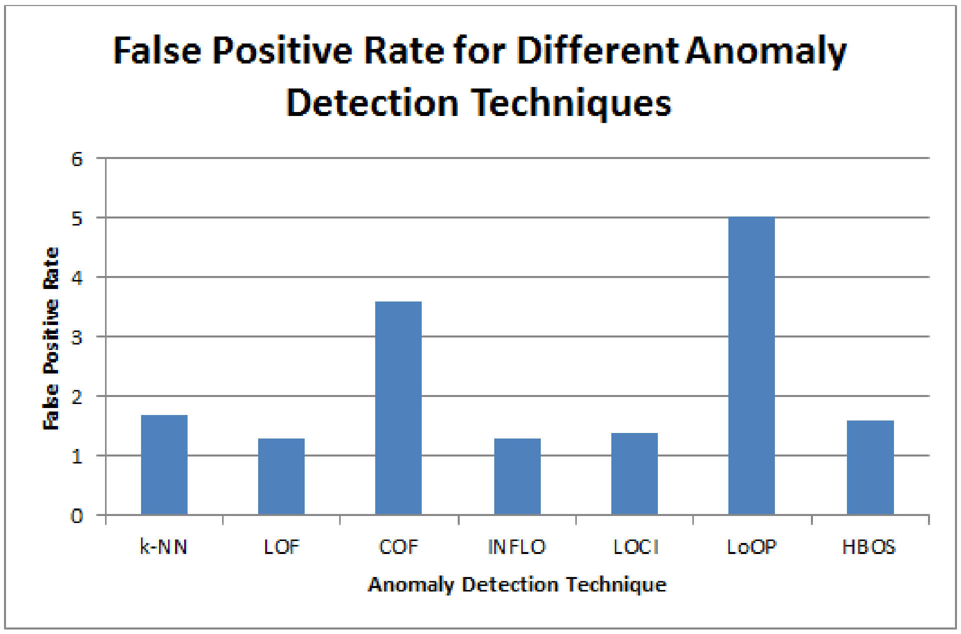


Figure 5.9: False Positive Rate for the seven anomaly detection techniques

Table 5.2: Performance results for the seven anomaly detection techniques

	% of FPR	% of TPR	% of $Accuracy$
k-NN	1.7%	100%	98.32%
LOF	1.3%	0%	97.63%
COF	3.6%	18.18%	95.55%
INFLO	1.3%	0%	97.63%
LOCI	1.4%	0%	97.53%
LoOP	5%	18.18%	94.16%
HBOS	1.6%	100%	98.42%

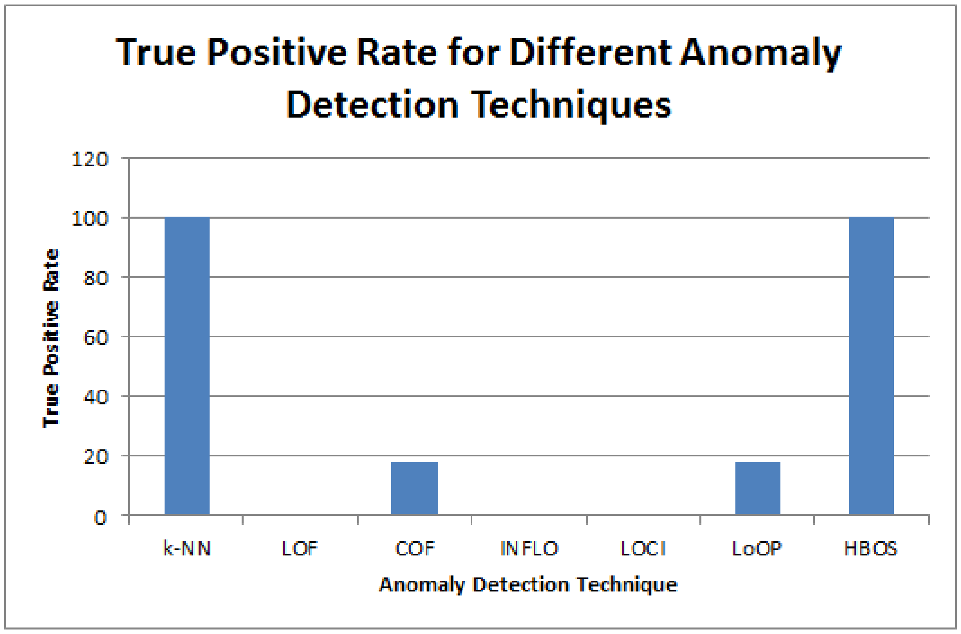


Figure 5.10: True Positive Rate for the seven anomaly detection techniques

anomaly detection techniques fail when attempting to solve a global anomaly detection problem, because they tend to look for local outliers. k-NN Global Anomaly Score and HBOS, on the other hand, are able to detect global outliers, but we expect them to perform poorly on a local anomaly detection problem [68].

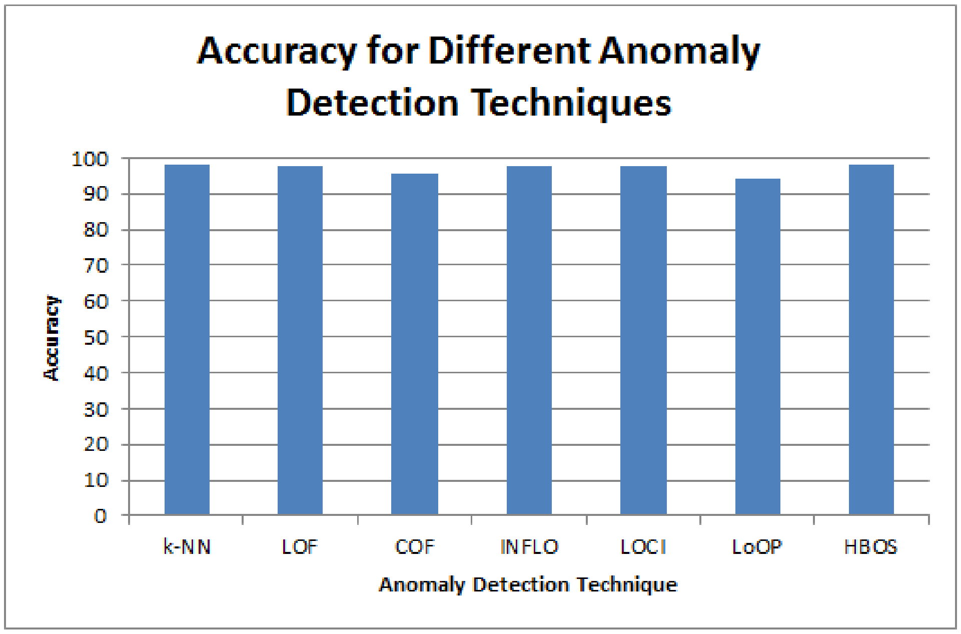


Figure 5.11: Accuracy for the seven anomaly detection techniques

5.3.2 Effectiveness for Diverse Tabnabbing Websites

In this section, we measure TabsGuard's effectiveness for diverse categories of target websites by examining its ability to detect tabnabbing attacks that target websites of different categories. As mentioned earlier in this thesis, we use 11 tabnabbing webpages. The tabnabbing pages that we created target three categories of web services: i) email services (*i.e.,* Gmail, Yahoo Mail, and Hotmail); ii) social networking websites (*i.e.*, Facebook, Linkedin, Twitter, and Pinterest); and iii) popular everyday-use websites (*i.e.,* Moodle, Wordpress, Wikipedia, and W3Schools).

We analyze the effectiveness of TabsGuard in correctly detecting tabnabbing pages that target websites of different categories. To this end, we run the seven anomaly detection operators in RapidMinor (*i.e., k*-NN, LOF, COF, INFLO, LOCI, LoOP, and HBOS) on the 11 tabnabbing pages in our dataset (*i.e.,* fake versions of Gmail, Yahoo Mail, Hotmail, Facebook, Linkedin, Twitter, Pinterest, Moodle, Wordpress, Wikipedia, and W3Schools) as these 11 pages are from three different categories of websites (*i.e.,* email services, social networking websites, and popular everyday-use websites). We record the outlier score determined by each anomaly detection technique for each one of the 11 pages and demonstrate the outlierness of each tabnabbing page for all the seven anomaly detection techniques in Figures 5.12, 5.13, 5.14, 5.15, 5.16, 5.17, and 5.18.

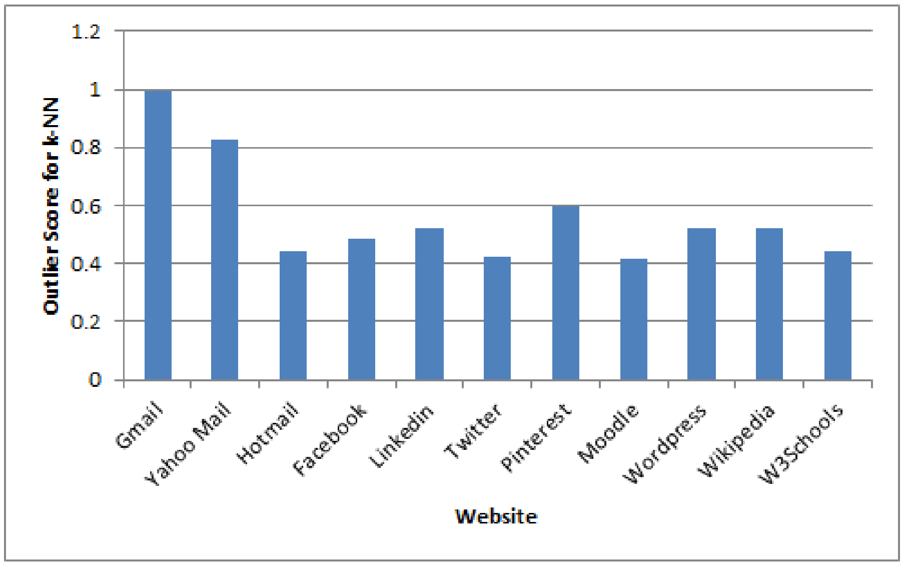


Figure 5.12: Outlierness of different tabnabbing websites measured by k-NN (Outlier score range for the whole dataset: 0.008$\pm$0.064)

With respect to the fact that the local outlier detection techniques (*i.e.,* LOF, COF, INFLO, LOCI, and LoOP) do not perform well in detecting the tabnabbing pages in our whole dataset, we observe that the other two anomaly detection techniques (*i.e.,* k-NN and HBOS) can successfully detect all 11 tabnabbing pages which

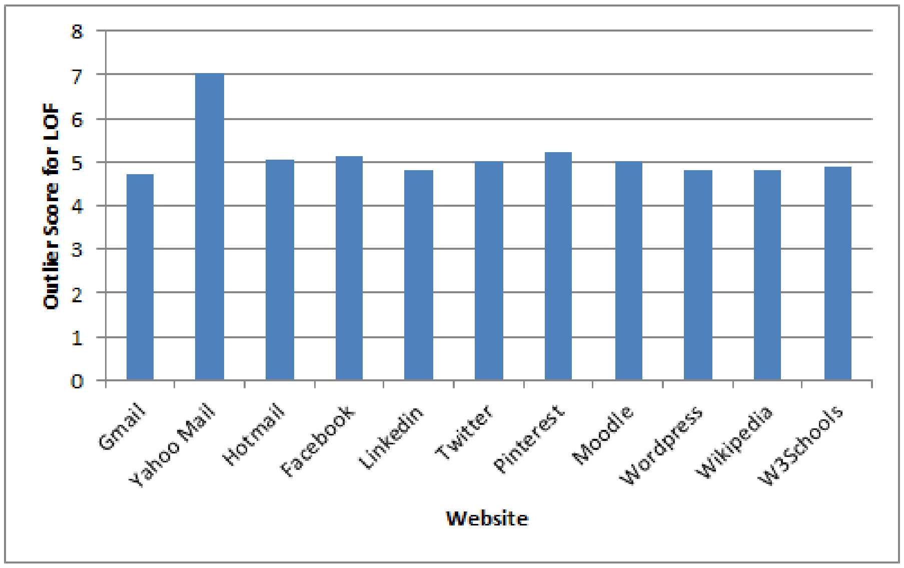


Figure 5.13: Outlierness of different tabnabbing websites measured by LOF (Outlier score range for the whole dataset: 8933.269$\pm$80692.960)

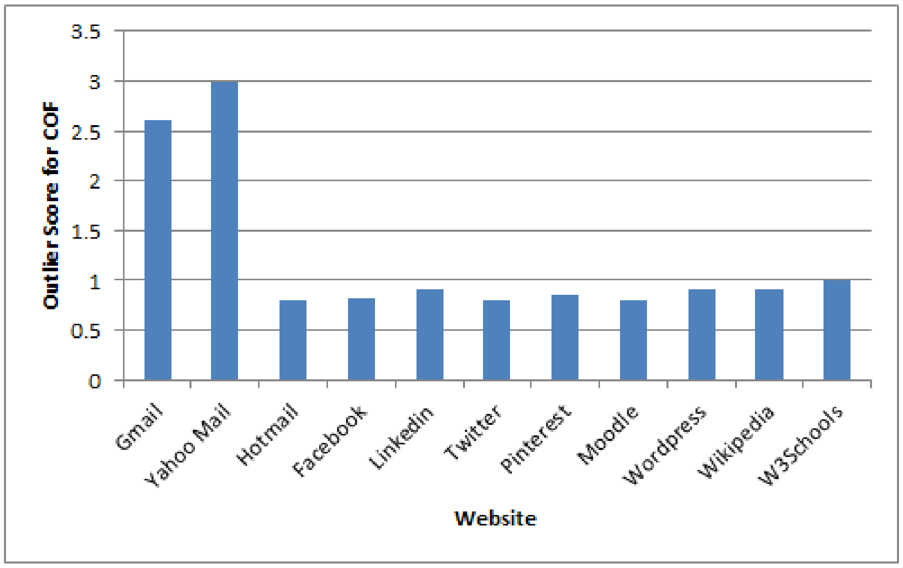


Figure 5.14: Outlierness of different tabnabbing websites measured by COF (Outlier score range for the whole dataset: 1.679$\pm$11.702)

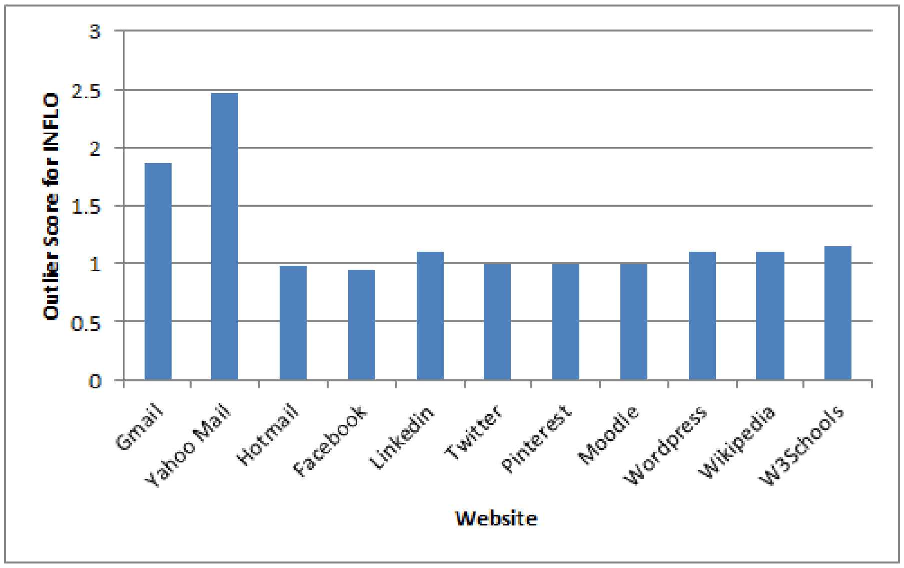


Figure 5.15: Outlierness of different tabnabbing websites measured by INFLO (Outlier score range for the whole dataset: 8920.857$\pm$80749.120)

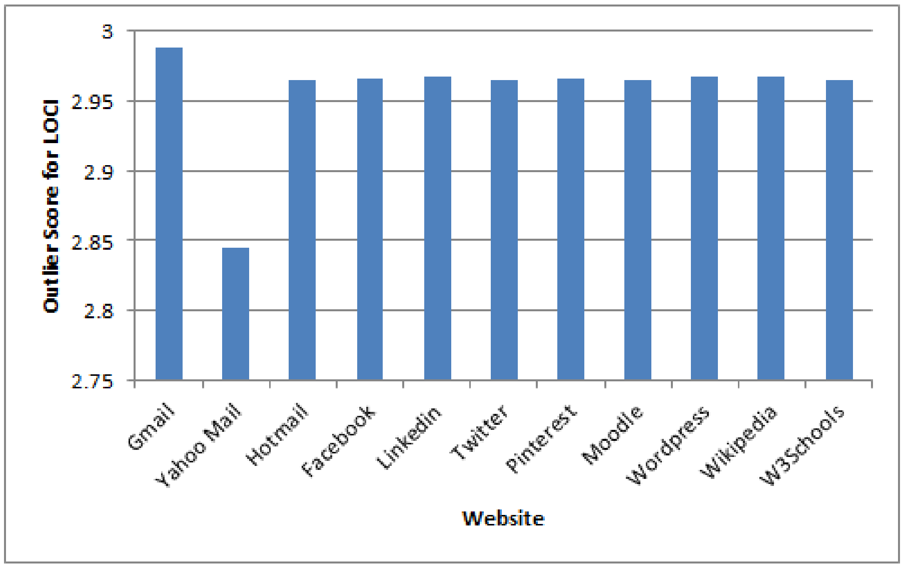


Figure 5.16: Outlierness of different tabnabbing websites measured by LOCI (Outlier score cut-off point: 3)

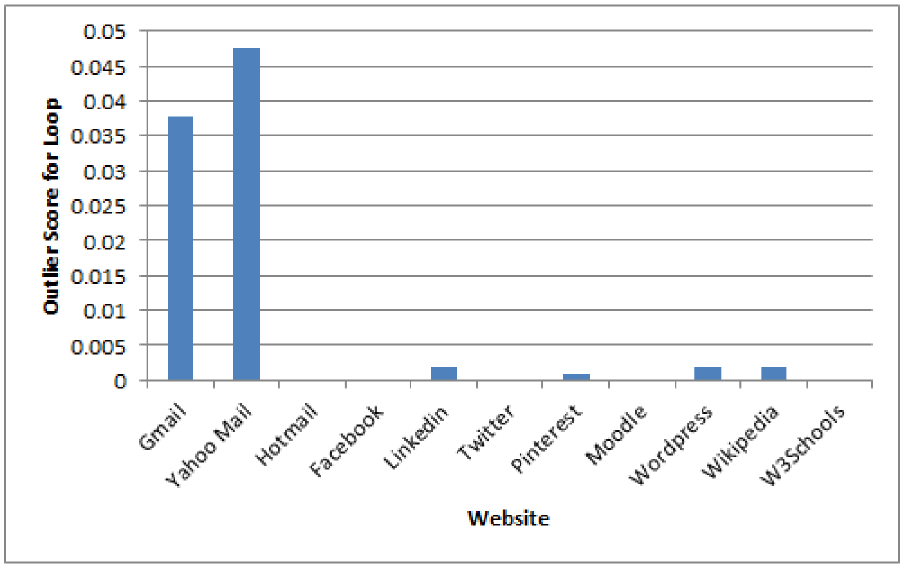


Figure 5.17: Outlierness of different tabnabbing websites measured by LoOP (Outlier score range for the whole dataset: 0.014$\pm$0.109)

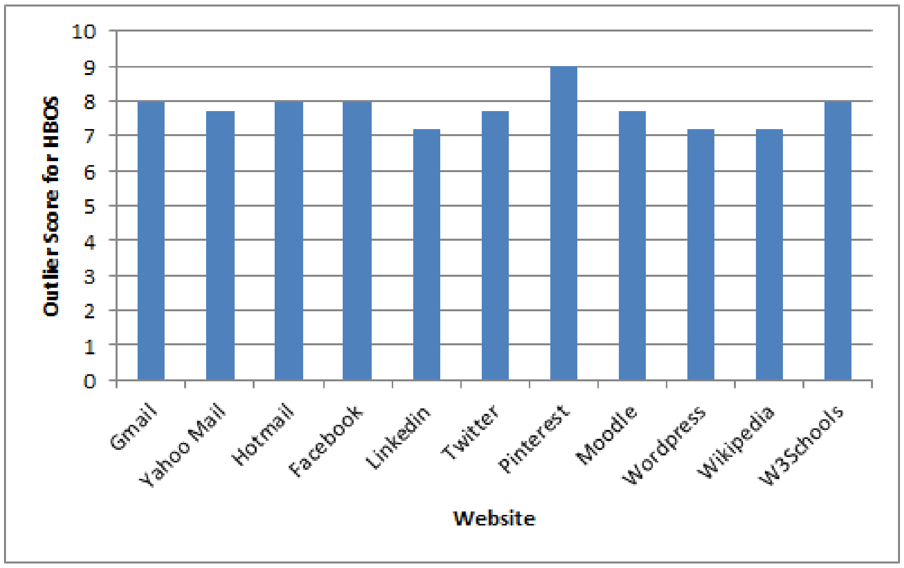


Figure 5.18: Outlierness of different tabnabbing websites measured by HBOS (Outlier score range for the whole dataset: 0.131$\pm$0.890)

are from three different categories of email services, social networking websites, and popular every-day use websites, respectively. According to our analysis, fake Gmail and fake Yahoo Mail are the only two websites that are even detected by some of the local anomaly detection techniques (*i.e.,* COF and LoOP), indicating that their degree of changes are more evident than those of the other 9 tabnabbing pages we have generated.

5.3.3 Overhead on Browser Startup and Page Load Time

In this section, we describe the experiments we conducted to evaluate TabsGuard's overhead as a Mozilla Firefox extension. We measure the overhead of TabsGuard on the browser with respect to two criteria: a) The impact of TabsGuard on browser startup time; and b) The impact of TabsGuard on the loading time of some websites. We measure this overhead for the three existing script-blocking Firefox extensions (*i.e.,* NoScript, Controle de Scripts, and YesScript) by conducting the same experiments for them as TabsGuard's and using the same software and hardware.

Evaluating the Impact of TabsGuard on Browser Startup Time

Browser extensions are loaded and executed whenever a new browser window is opened. In other words, a browser extension can have an impact on the time it takes for the browser to start up every time a window is opened [82]. Thus, we believe that the impact of our extension on the browser startup time can be a good indicator of the overhead our extension imposes on the browser. In order to measure the overhead of TabsGuard, we use the *About Startup* Firefox extension [76]. Using

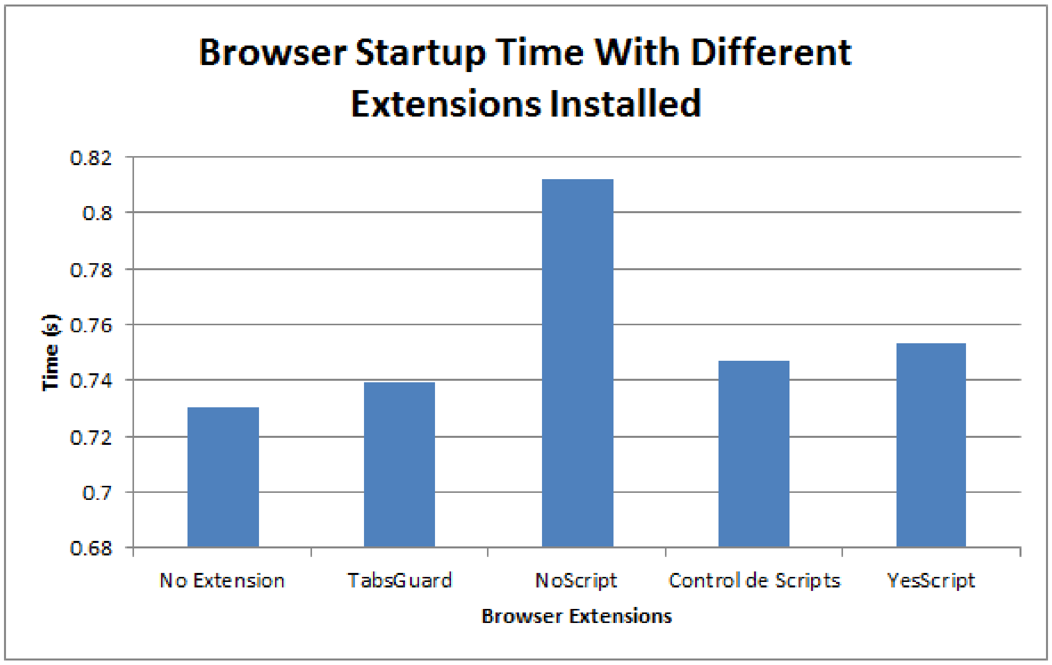


Figure 5.19: The impact of different anti-tabnabbing browser extensions on browser startup time

this extension, we measure the average startup time of Firefox with and without having TabsGuard installed. For the sake of comparison, we also measure the startup time using the three existing script-blocking Firefox extensions.

The *About Startup* extension for Firefox can be used to measure startup performance as it gathers and displays startup log history. To run our experiment using this extension, we first set up a new Firefox profile with no extensions installed on it. We then install *About Startup* and open and quit Firefox repeatedly for 20 consecutive times. The reason we repeat this task for 20 times is to make this test match Talos [83]. Talos is a performance testing framework used by Mozilla. We then obtain the results from the `about:startup` page. We repeat this experiment with a) *About*

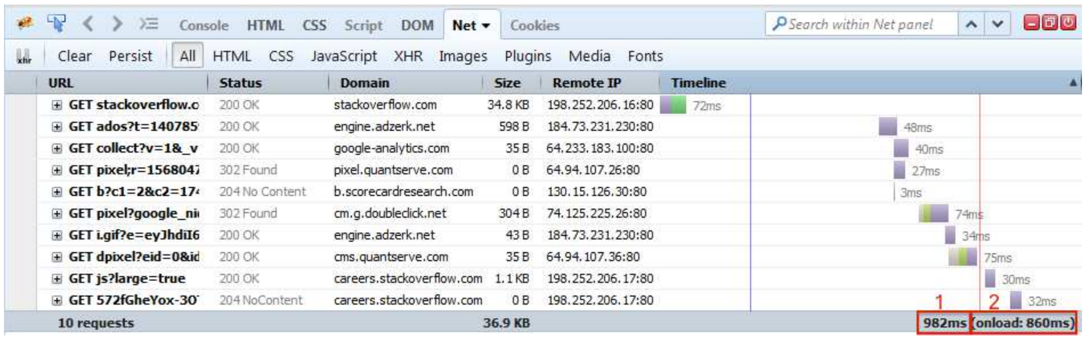
Startup and TabsGuard, b) *About Startup* and NoScript, c) *About Startup* and Controle de Script, and d) *About Startup* and YesScript installed on Firefox. Among the time measurements provided by *About Startup*, we make use of `firstPaint` [84] as it is almost similar to what Talos tests. `firstPaint` is a startup milestone showing the time when the first Firefox window becomes visible to the user. It should be noted that `firstPaint` can vary on different systems [76]. Therefore, to ensure consistency, we conduct all our experiments on the same system.

For the four cases where each four anti-tabnabbing extensions (including TabsGuard) are installed on Firefox plus the case where no extension except for *About Startup* is installed on Firefox, we get the `firstPaint` values for the 20 runs and calculate the average of this number. Figure 5.19 illustrates the results we obtain for each case. As shown in this figure, TabsGuard does not impose a significant overhead on Firefox startup time. In fact, the experiments demonstrate that TabsGuard's startup performance is better than those of NoScript, Controle de Scripts, and YesScript as TabsGuard makes a smaller delay time on the browser's startup regarding the baseline case where no extension is installed on the browser.

To further investigate TabsGuard's impact on the browser startup time, we use the *Wilcoxon Rank-Sum test* to compare the 20 values we obtained for the `firstPaint` milestone when our extension is installed on the browser with the 20 corresponding values obtained for the case where no extension is intalled on the browser. The Wilcoxon Rank-Sum test is a non-parametric statistical test that assesses whether one set of data tends to have larger values than another set or not — considering that the two sets are independent. This test aims at answering whether two sets of data are really different or not. We report the result of this test in terms of the estimated probability of rejecting a hypothesis of “no difference” (using the p-value). The p-value is used to indicate a probability calculated after a test/study on different sets of data. If the obtained p-value for a test is less than a predetermined significance level, which is often 0.05 or 0.01, there is a significant difference between the two sets of data [85]. Using the Wilcoxon Rank-Sum test, we get p-value > 0.2 which confirms there is no significant difference between the 20 values for browser startup time obtained for an extension-free browser and the 20 values obtained for a browser with our extension installed on it. That is to say, this test and the resulted p-value illustrates that TabsGuard does not have any significant impact on the browser startup time.

Evaluating the Impact of TabsGuard on Page Load Time

We measure the impact of TabsGuard on the execution time of the browser and compare it with those of NoScript, Controle de Scripts, and YesScript using the `onload` time of eight different websites (*i.e.,* Google, Facebook, Yahoo, Linkedin, Twitter, Moodle, Wordpress, and Gmail). The `onload` time is the time elapsed since the start of the first request until the load event is fired by the page and finished executing [86]. We use Firebug’s Net panel [79, 86, 87] to calculate this time for some of the popular web applications. As depicted in Figure 5.20, Firebug shows two numbers representing two different approaches to the way we can measure the page load time: (1) the total time needed to load all requests (982ms in Figure 5.20) and (2) the time when the onload event occurred (860ms in Figure 5.20). However, for the websites that use dynamic refreshing of the contents of the page (*e.g.,* Facebook, Linkedin, Twitter, etc.), new requests continue to show up even after the `onload`



URL	Status	Domain	Size	Remote IP	Timeline
GET stackoverflow.c	200 OK	stackoverflow.com	34.8 KB	198.252.206.16:80	72ms
GET ados?t=140785	200 OK	engine.adzerk.net	598 B	184.73.231.230:80	48ms
GET collect?v=1&_v	200 OK	google-analytics.com	35 B	64.233.183.100:80	40ms
GET pixel;r=1568047	302 Found	pixel.quantserve.com	0 B	64.94.107.26:80	27ms
GET b?c1=2&c2=17	204 No Content	b.scorecardresearch.com	0 B	130.15.126.30:80	3ms
GET pixel?google_ni	302 Found	cm.g.doubleclick.net	304 B	74.125.225.26:80	74ms
GET i.gif?e=eyJhdiI6	200 OK	engine.adzerk.net	43 B	184.73.231.230:80	34ms
GET dpixel?eid=0&id	200 OK	cms.quantserve.com	35 B	64.94.107.36:80	75ms
GET js?large=true	200 OK	careers.stackoverflow.com	1.1 KB	198.252.206.17:80	30ms
GET 572fGheYox-30	204 NoContent	careers.stackoverflow.com	0 B	198.252.206.17:80	32ms
10 requests			36.9 KB		982ms (onload: 860ms)

Figure 5.20: Firebug's Net panel for measuring page load time

event has finished executing; hence, the total time needed to load all requests increases over time. This is the reason why we choose the `onload` time as our measure for the page load time. We obtain this measure for the eight abovementioned websites for the case where there is no extension installed on Firefox as well as the cases where TabsGuard, NoScript, Controle de Scripts, and YesScript are installed on Firefox. We are aware that network speed could influence the `onload` time of a website. Thus, we conduct the experiments for all websites and in all five cases using the same network connection and at roughly the same time. Figure 5.21 shows the results for this set of experiments. As demonstrated, TabsGuard imposes no more overhead on the loading time of these eight websites that the other three extensions. In fact, in most cases its overhead is among the least.

5.4 A Comparative Analysis

In this section, we present a case-study in which we compare the performance of TabsGuard with the performance of TabShots [8] — the only existing technique that reports quantitative tabnabbing detection results. This comparison is based on the

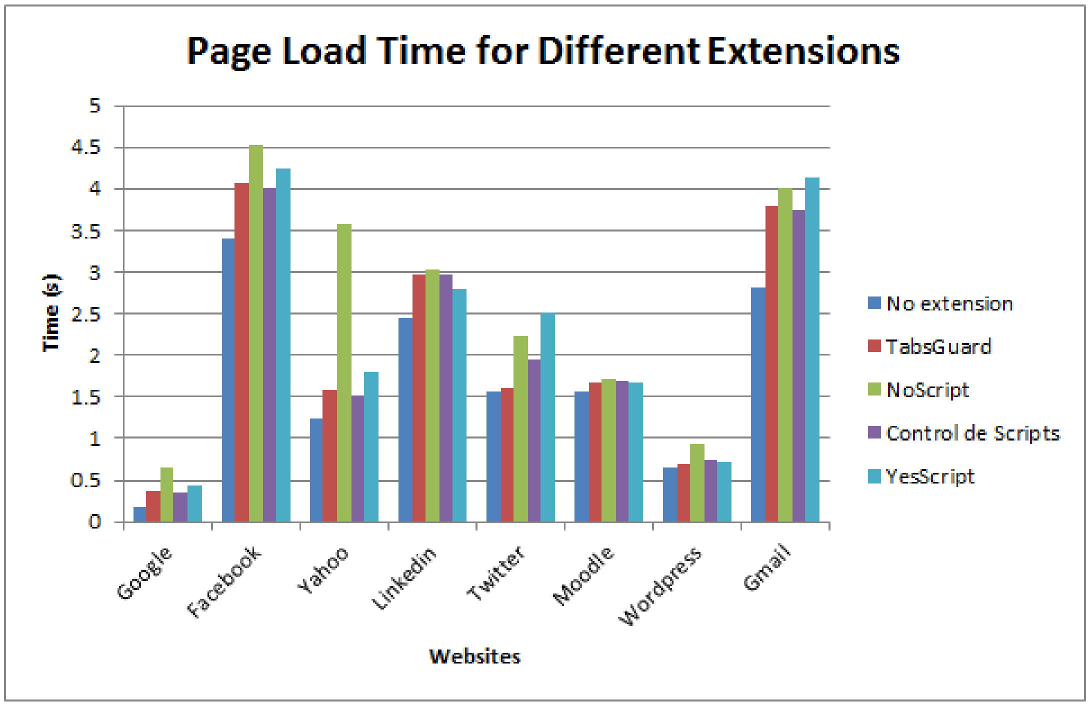


Figure 5.21: The impact of different anti-tabnabbing browser extensions on page load time

detection reports of TabShots' results for the top 20 websites of Alexa. Using the same websites, we report the results of our approach obtained when k-NN is used as the anomaly detection technique.

Table 5.3 shows the top 20 Alexa websites and their percentage of changed blocks. Column 2 shows the results of TabShots as reported in [8] and Columns 3-5 show the results obtained by TabsGuard. Since our metrics show similarity rate rather than the percentage of change, we normalize the values of $H1$, $H2$, and $H3$ accordingly. To avoid any confusion, we rename the normalized versions of the metrics to $H1_2$, $H2_2$, and $H3_2$. Table 5.3 clearly shows that the mechanism we utilize for detecting changes in a page is superior to that of TabShots. The reason is that TabsGuard is able to detect minor changes in legitimate websites (such as the appearance of

Table 5.3: Comparison of TabsGuard and TabShots using top 20 Alexa websites.

Website	**% of Changed Blocks by TabShots**	**% of Changes by TabsGuard** $H1_2$	$H2_2$	$H3_2$
facebook.com	0.38	0.00	0.00	1E-07
google.com	0.00	0.00	0.00	0.00
youtube.com	4.05	0.00	0.00	0.00
yahoo.com	5.31	0.00	0.00	0.00
baidu.com	0.00	0.00	0.00	0.00
wikipedia.org	0.73	0.00	0.00	0.00
live.com	2.65	0.00	0.0015	6.2E-06
twitter.com	2.91	0.00	0.00	0.00
qq.com	6.00	0.00	0.00	0.00
amazon.com	2.57	0.00	0.00	0.00
blogspot.com	0.32	0.00	0.00	0.00
linkedin.com	0.26	0.00	0.00	0.00
taobao.com	0.49	0.00	0.00	2.59E-05
google.co.in	0.00	0.00	0.00	0.00
yahoo.co.jp	4.13	0.00	0.00	0.00
sina.com.cn	1.24	0.00	0.00	0.00
msn.com	23.22	0.00	0.01	0.00
google.com.hk	0.00	0.00	0.00	0.00
google.de	0.00	0.00	0.00	0.00
bing.com	0.00	0.00	0.00	0.00

a new email or a new comment under a post in a social networking website) and recognize them as benign changes. It should also be noted that TabShots measures the percentage of changed blocks (*i.e.,* 10x10 tiles in the screenshot of a webpage), whereas TabsGuard measures the changes across the whole webpage with respect to the tree structure of the page. This is a distinct advantage of TabsGuard over TabShots, because measuring the changes block by block might lead to missing some changes and this could result in more false negatives.

Table 5.4: Comparison of TabsGuard and TabShots in worst-case scenarios.

Website	**% of Changed Blocks by TabShots**	**% of Changes by TabsGuard** H_1	H_2	H_3
americanexpress.com	38.93	0.00	0.00	0.00
mlb.com	97.31	0.00	0.0009237	1.9E-06

The authors of TabShots also give two examples of their worst-case scenario for the websites with changes between 5% and 40% and the websites with changes more than 40%. These two examples are `americanexpress.com` and `mlb.com` with the reported results of 38.93% and 97.31% for the changed blocks, respectively. The high percentage of changed blocks for the former website is claimed to be due to the delayed loading of a background image. In our experiment, no such case has been observed with TabsGuard. The reason is that TabsGuard waits for each page to be fully loaded before it starts functioning. As shown in Table 5.4, we detect very small changes for these two websites.

5.5 Relevance Values for the Heuristic-Based Metrics

To recognize the impact of each one of the heuristic-based metrics presented by us (Section 4.3.1) on a page being tabnabbing or legitimate, we conduct an analysis on these metrics using RapidMiner. Given Equation 5.4, where $H1$ (*i.e.,* CP), $H2$ (*i.e.,* Cosine similarity and class names), $H3$ (*i.e.,* TFDA), $H4$ (*i.e.,* `<input>` elements added to the page), and $H5$ (*i.e.,* iframes in the page before and after tab switch) are our heuristic-based metrics that are the attributes in our complete dataset, and Y is the result indicating whether each page in our dataset is legitimate or tabnabbing, we are interested in finding values for relevance coefficients a, b, c, d, and e.

$$Y = aH1 + bH2 + cH3 + dH4 + eH5 \tag{5.4}$$

Using the Weight by Correlation operator in RapidMiner, we obtain the values shown in Table 5.5 for a, b, c, d, and e as a result of which we can rewrite Equation 5.4 as Equation 5.5. The Weight by Correlation operator calculates the weight of attributes $H1$ to $H5$ with respect to the label attribute (Tabnabbing) by computing the value of correlation for each attribute. The higher the weight of an attribute, the more relevant it is considered to result Y.

Table 5.5: Relevance values computed for the five proposed metrics.

Relevance Coefficient	**Value**
a	1.000
b	0.927
c	0.743
d	0.952
e	0.000

$$Y = H1 + 0.927H2 + 0.743H3 + 0.952H4 \tag{5.5}$$

As shown in Table 5.5, we can infer that $H1$ (*i.e.,* CP), $H4$ (*i.e.,* `<input>` elements added to the page), and $H2$ (*i.e.,* Cosine similarity and class names) are the most relevant attributes to the determined value for Y, whereas the relevance of $H3$ (*i.e.,* TFDA) is a little less. The zero value obtained for $H5$ conveys that the existence of iframes in the page before and after tab switch is not relevant to a webpage being detected as tabnabbing. The reason is that nowadays, the majority of websites, especially the popular ones such as Google and Facebook use iframes for embedding advertisements or other non-malicious content within their HTML code. Therefore, the existence of iframes cannot be considered as a characteristic of malicious webpages.

5.6 Summary

In this chapter, we described the evaluation of our proposed tabnabbing detection technique from three points of view. First, we conducted a set of experiments to measure the false positive rate, true positive rate, and accuracy that TabsGuard obtains using each of the seven anomaly detection techniques introduced earlier. Second, we measured the ability of TabsGuard to detect tabnabbing websites of different categories. Third, we evaluated the performance of TabsGuard as a Firefox extension by measuring its impact on the browser startup time and the page load time. We also compared the results of these experiments with the results we obtained for the three existing script-blocking Firefox extensions. Moreover, we compared TabsGuard's detection results when using k-NN as the outlier detection technique with the detection results of an existing tabnabbing detection technique (*i.e.,* TabShots) for a small portion of legitimate data. Finally, we computed the relevance values for the five heuristic-based metrics that we proposed.

According to the definition of the tabnabbing attack, a technique that could detect the changes made to the title, favicon, and appearance of a webpage while it was inactive could in fact detect the attack. In our case, the detection of all tabnabbing pages is ensured by the way TabsGuard has been designed, because our approach of keeping track of all major changes made to the title, favicon, and HTML layout of a page cannot miss any tabnabbing attack of either variant. If among title, favicon, and layout, one or two of the three do not change, the attack is still detected by TabsGuard, not as a tabnabbing attack, but as a classical phishing attack.

Moreover, regarding the evaluation results, we observed that local outlier detection techniques do not work well when solving a global outlier detection problem, that in our case is finding tabnabbing pages among the whole dataset of legitimate and tabnabbing pages. Therefore, TabsGuard can potentially guarantee zero false negative rate (*i.e.,* 100% detection rate) in case k-NN or HBOS are used for detecting the outliers.

Finally, the performance experiments indicated that TabsGuard imposes no significant overhead on the browser compared to the three script-blocking Firefox extensions. Furthermore, our approach is effective enough to detect tabnabbing attacks that target trusted websites of different categories when using k-NN or HBOS as the outlier detection technique.

Chapter 6

Conclusion

6.1 Summary

Phishing is one of the dominant attacks to exploit users' confidential information. While most of the existing anti-phishing techniques are still struggling with effectively detecting and preventing phishing attacks, a new variant of phishing — named tabnabbing — has been identified. Tabnabbing is a more sophisticated way of tricking users into giving away their personal information and credentials by leveraging the facilities tabs offer to modern Web browsers. So far, two variants of tabnabbing attacks have been reported: script-based and script-free. The script-based variant can be tackled using the existing script-blocking browser extensions, whereas the script-free variant is more complicated to detect. Despite the necessity of effectively preventing tabnabbing attacks, few anti-tabnabbing tools and techniques have been proposed, most of which are not capable of preventing both variants of the attack.

To address the above problem, we have proposed TabsGuard, a hybrid tabnabbing detection and prevention approach that is able to detect both variants of the tabnabbing attack. Our tabnabbing detection approach combines the benefits of five heuristic-based metrics and seven anomaly detection techniques to detect tabnabbing attacks with respect to the changes made to the structure of a webpage while it is out of focus. After detecting the attack on a browser, our approach can actively prevent it by blacklisting the webpage and redirecting the user to a trusted domain.

We have developed TabsGuard as a browser extension for Mozilla Firefox and have evaluated it from both functionality and performance perspectives. Our evaluation results demonstrated TabsGuard's capability of correctly detecting all tabnabbing attacks when k-NN and HBOS are used as the anomaly detection technique analyzing the degree of changes made to a webpage. We also demonstrated that TabsGuard does not incur a significant overhead on the browser's startup time and page load time, and it is able to detect tabnabbing attacks that target some examples of web applications from different categories (*e.g.*, email services, social networking websites, and popular everyday-use websites).

6.2 Limitations and Future Work

As part of our future work, we plan to extend TabsGuard and the underlying implementation in a number of ways as described in the following paragraphs.

The number of tabnabbing pages we currently have at hand is sufficient for being used in unsupervised anomaly detection techniques, but it cannot match the large number of the available legitimate pages and therefore cannot be used to build a training dataset. Regarding this challenge, we plan to generate more tabnabbing pages to be able to build a training dataset and investigate and compare the effectiveness of different supervised classification techniques for detecting tabnabbing attacks. This will also enable us to detect future tabnabbing attacks.

We can incorporate more heuristic-based metrics into our tabnabbing detection approach to improve the accuracy of measuring the degree of changes occurred to a webpage during the time it is out of focus.

We can further develop our approach in such a way that it could detect tabnabbing attacks on the fly. To this end, we should embed the implementation of the analysis of the changes that are made to the structure of the page into the extension, for instance by including a JavaScript implementation of k-NN and HBOS anomaly detection algorithms in the implementation of the extension. In this way, all the tabnabbing detection steps could be performed at runtime.

We can think of more sophisticated tabnabbing attack models and make an attempt to modify our approach in a way that it could defend against such attacks. For instance, a suggested attack model could cause the browser to crash upon opening a malicious page in a tab. It would then reload the malicious tab with the appearance of a trusted website once the browser restores the tabs after the crash. This will make the attack more likely to succeed, because there will be an even lower chance of the attack being detected by the user.

We can extend our approach in such a way that it detects classical phishing attacks in addition to tabnabbing attacks. For this purpose, we should include phishing detection-specific approaches (*e.g.,* adding heuristics that detect fraudulent URLs, conducting text analyses, etc.) in our extension. The new approach may also allow for an earlier detection of tabnabbing attacks if the page before tab switch could be identified as malicious with respect to certain characteristics of phishing pages.

Finally, we plan to extend TabsGuard to other Web browsers and make it available as a service for public use.

Bibliography

[1] Sun Bin, Wen Qiaoyan, and Liang Xiaoying. A DNS-Based Anti-Phishing Approach. In *Proceedings of the 2010 Second International Conference on Networks Security, Wireless Communications and Trusted Computing - Volume 02*, NSWCTC '10, pages 262–265, Washington, DC, USA, 2010. IEEE Computer Society.

[2] Pawan Prakash, Manish Kumar, Ramana Rao Kompella, and Minaxi Gupta. PhishNet: Predictive Blacklisting to Detect Phishing Attacks. In *Proceedings of the 29th Conference on Information Communications*, INFOCOM'10, pages 346–350, Piscataway, NJ, USA, 2010. IEEE Press.

[3] Anti-Phishing Working Group. Global Phishing Survey: Trends and Domain Name Use in 2H2013. `http://docs.apwg.org/reports/APWG_GlobalPhishingSurvey_2H2013.pdf`, June 2014.

[4] A. Belabed, E. Aïmeur, and A. Chikh. A Personalized Whitelist Approach for Phishing Webpage Detection. In *Proceedings of the 2012 Seventh International Conference on Availability, Reliability and Security*, ARES '12, pages 249–254, Washington, DC, USA, 2012. IEEE Computer Society.

[5] Matthew Dunlop, Stephen Groat, and D. Shelly. Goldphish: Using Images for Content-Based Phishing Analysis. In *Internet Monitoring and Protection (ICIMP), 2010 Fifth International Conference on*, pages 123–128, May 2010.

[6] Federico Maggi. Are the Con Artists Back? A Preliminary Analysis of Modern Phone Frauds. In *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, pages 824–831. IEEE, 2010.

[7] Aza Raskin. Tabnabbing: A New Type of Phishing Attack. `http://www.azarask.in/blog/post/a-new-type-of-phishing-attack/`, June 2014.

[8] Philippe De Ryck, Nick Nikiforakis, Lieven Desmet, and Wouter Joosen. TabShots: Client-Side Detection of Tabnabbing Attacks. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ASIA CCS '13, pages 447–456, New York, NY, USA, 2013. ACM.

[9] Krebs on Security. Devious New Phishing Tactic Targets Tabs. `http://avivraff.com/research/phish/article.php?1464682399`, June 2014.

[10] John K. Ousterhout. Scripting: Higher Level Programming for the 21st Century. *Computer*, 31(3):23–30, Mar 1998.

[11] InformAction Open Source Software. Noscript. `http://noscript.net/`, June 2014.

[12] Mozilla Foundation. Controle de Scripts. `https://addons.mozilla.org/en-US/firefox/addon/controle-de-scripts/`, June 2014.

[13] Chrome Web Store. Script Block. `https://chrome.google.com/webstore/detail/scriptblock/hcdjknjpbnhdoabbngpmfekaecnpajba?hl=en`, June 2014.

[14] Alexa. Alexa - Actionable Analytics for the Web. `http://www.alexa.com/`, May 2014.

[15] Peter E Duda and Richard O Hart. Pattern Classification and Scene Analysis, 1973.

[16] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. LOF: Identifying Density-Based Local Outliers. In *ACM Sigmod Record*, volume 29, pages 93–104. ACM, 2000.

[17] Jian Tang, Zhixiang Chen, Ada Wai-Chee Fu, and David W Cheung. Enhancing Effectiveness of Outlier Detections for Low Density Patterns. In *Advances in Knowledge Discovery and Data Mining*, pages 535–548. Springer, 2002.

[18] Mennatallah Amer, Markus Goldstein, and Slim Abdennadher. Enhancing One-Class Support Vector Machines for Unsupervised Anomaly Detection. In *Proceedings of the ACM SIGKDD Workshop on Outlier Detection and Description*, pages 8–15. ACM, 2013.

[19] H. Orman. The Compleat Story of Phish. *Internet Computing, IEEE*, 17(1):87–91, Jan 2013.

[20] Carine G. Webber, Maria de Ftima W. do Prado Lima, and Felipe S. Hepp. Testing Phishing Detection Criteria and Methods. In Sabo Sambath and Egui Zhu,

editors, *Frontiers in Computer Education*, volume 133 of *Advances in Intelligent and Soft Computing*, pages 853–858. Springer Berlin Heidelberg, 2012.

[21] Intuit. Phishing, Pharming, Vishing and Smishing. `https://security.intuit.com/phishing.html`, July 2014.

[22] Anuj Kumar Jain. User-Definable Images in Bookmarks, 2000.

[23] Ben Vinegar and Anton Kovalyov. *Third-Party JavaScript.* Manning Publications Co., 2013.

[24] Adblock Plus. Third-Party JavaScript - Yes, it is a security risk. `https://adblockplus.org/blog/third-party-javascript-yes-it-is-a-security-risk`, July 2014.

[25] John Arana. Introducing Flash Widgets. *Creating Flash Widgets with Flash CS4 and ActionScript 3.0*, pages 1–4, 2008.

[26] Learn How To Hack - Best Online Ethical Hacking Website. Advanced Tabnabbing Tutorial. `http://www.hackingloops.com/2012/04/advanced-tabnabbing-tutorial.html`, June 2014.

[27] W3Schools. HTML iframe Tag. `http://www.w3schools.com/tags/tag_iframe.asp`, July 2014.

[28] Google Caja. Browser History Mining. `https://code.google.com/p/google-caja/wiki/HistoryMining`, August 2014.

[29] The Open Web Application Security Project (OWASP). Cross-site Scripting (XSS). `https://www.owasp.org/index.php/Cross-site_Scripting_%28XSS%29`, August 2014.

[30] G.A Di Lucca, AR. Fasolino, M. Mastoianni, and P. Tramontana. Identifying Cross Site Scripting Vulnerabilities in Web Applications. In *Telecommunications Energy Conference, 2004. INTELEC 2004. 26th Annual International*, pages 71–80, Sept 2004.

[31] Unicode. About the Unicode Standard. `http://www.unicode.org/standard/standard.html`, July 2014.

[32] Gizmodo. How Non-Latin Domain Names Could Be Used to Steal Your Money. `http://gizmodo.com/5439471/how-non+latin-domain-names-could-be-used-to-steal-your-money`, July 2014.

[33] Gaurav Gupta and Josef Pieprzyk. Socio-Technological Phishing Prevention. *Information Security Technical Report*, 16(2):67–73, May 2011.

[34] Yue Zhang, Jason I. Hong, and Lorrie 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.

[35] Christian Ludl, Sean McAllister, Engin Kirda, and Christopher Kruegel. On the Effectiveness of Techniques to Detect Phishing Sites. In Bernhard M. Hmmerli

and Robin Sommer, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment*, volume 4579 of *Lecture Notes in Computer Science*, pages 20–39. Springer Berlin Heidelberg, 2007.

[36] PhishTank. PhishTank - Out of the Net, into the Tank. `http://www.phishtank.com/`, July 2014.

[37] M. Khonji, Y. Iraqi, and A Jones. Phishing Detection: A Literature Survey. *Communications Surveys Tutorials, IEEE*, 15(4):2091–2121, Fourth 2013.

[38] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986.

[39] Mozilla Corporation. Mozilla Persona. `https://login.persona.org/about`, June 2014.

[40] Benjamin Smedberg. Don't Use Mozilla Persona to Secure High-Value Data. `http://benjamin.smedbergs.us/blog/2014-02-11/dont-use-mozilla-persona-to-secure-high-value-data/`, August 2014.

[41] Mozilla Foundation. YesScript. `https://addons.mozilla.org/en-US/firefox/addon/yesscript/`, June 2014.

[42] Chrome Web Store. NotScripts. `https://chrome.google.com/webstore/detail/notscripts/odjhifogjcknibkahlpidmdajjpkkcfn?hl=en`, June 2014.

[43] Chrome Web Store. ScriptSafe. `https://chrome.google.com/webstore/detail/scriptsafe/oiigbmnaadbkfbmpbfijlflahbdbdgdf?hl=en`, June 2014.

[44] Chrome Web Store. Script Defender. `https://chrome.google.com/webstore/detail/scriptdefender/celgmkbkgakmkfboolifhbllkfiepcae?hl=en`, June 2014.

[45] S.A. Unlu and K. Bicakci. NoTabNab: Protection Against the Tabnabbing Attack. In *eCrime Researchers Summit (eCrime), 2010*, pages 1–5, Oct 2010.

[46] R. K. Suri, D. S. Tomar, and D. R. Sahu. An Approach to Perceive Tabnabbing Attack. In *International Journal of Scientific & Technology Research*, volume 1. 2012.

[47] S Sarika and Varghese Paul. An Anti-Phishing Framework to Defend Tabnabbing Attack. In *International Conference on Security and Authentication*, pages 132–135, March 2014.

[48] S Sarika and Varghese Paul. Distributed Software Agents for Antiphishing. *International Journal of Computer Science Issues (IJCSI)*, 10(3), May 2013.

[49] Nielsen Norman Group. How Long Do Users Stay on Web Pages? `http://www.nngroup.com/articles/how-long-do-users-stay-on-web-pages/`, June 2014.

[50] RapidMiner. RapidMiner. `http://rapidminer.com/`, July 2014.

[51] Mozilla Foundation. Document Object Model (DOM). `https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction`, July 2014.

[52] W3C. Document Object Model (HTML) Level 1. `http://www.w3.org/TR/REC-DOM-Level-1/level-one-html.html`, August 2014.

[53] Jonathan Robie and Texcel Research. What is the Document Object Model? `http://www.w3.org/TR/WD-DOM/introduction.html`, July 2014.

[54] Angelo P E Rosiello, E. Kirda, C. Kruegel, and F. Ferrandi. A Layout-Similarity-Based Approach for Detecting Phishing Pages. In *Security and Privacy in Communications Networks and the Workshops, 2007. SecureComm 2007. Third International Conference on*, pages 454–463, Sept 2007.

[55] Filippo Ricca and Paolo Tonella. Analysis and Testing of Web Applications. In *Proceedings of the 23rd International Conference on Software Engineering*, ICSE '01, pages 25–34, Washington, DC, USA, 2001. IEEE Computer Society.

[56] Paolo Tonella and Filippo Ricca. Statistical Testing of Web Applications. *Journal of Software Maintenance and Evolution: Research and Practice*, 16(1-2):103–127, 2004.

[57] Thomas Gottron. Clustering Template-Based Web Documents. In *Proceedings of the IR Research, 30th European Conference on Advances in Information Retrieval*, ECIR'08, pages 40–51, Berlin, Heidelberg, 2008. Springer-Verlag.

[58] E Garcia. Cosine Similarity and Term Weight Tutorial. *Information Retrieval Intelligence*, 2006.

[59] Isabel F. Cruz, Slava Borisov, Michael A. Marks, and Timothy R. Webb. Measuring Structural Similarity Among Web Documents: Preliminary Results. In Roger D. Hersch, Jacques Andr, and Heather Brown, editors, *Electronic Publishing, Artistic Imaging, and Digital Typography*, volume 1375 of *Lecture Notes in Computer Science*, pages 513–524. Springer Berlin Heidelberg, 1998.

[60] Anastasios Tombros and Zeeshan Ali. Factors Affecting Web Page Similarity. In *Proceedings of the 27th European Conference on Advances in Information Retrieval Research*, ECIR'05, pages 487–501, Berlin, Heidelberg, 2005. Springer-Verlag.

[61] Tatiana A. Starikovskaya Maxim A. Babenko. Computing Longest Common Substrings Using Suffix Arrays. `http://lpcs.math.msu.su/~pritykin/csr2008presentations/starikovskaya.pdf`, July 2014.

[62] TopCoder. Dynamic Programming: From Novice to Advanced. `http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=dynProg`, July 2014.

[63] Oracle Data Mining Concepts. Anomaly Detection. `http://docs.oracle.com/cd/B28359_01/datamine.111/b28129/anomalies.htm#DMCON006`, June 2014.

[64] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly Detection: A Survey. *ACM Computing Surveys (CSUR)*, 41(3):15, 2009.

[65] Wen Jin, Anthony KH Tung, Jiawei Han, and Wei Wang. Ranking Outliers Using Symmetric Neighborhood Relationship. In *Advances in Knowledge Discovery and Data Mining*, pages 577–593. Springer, 2006.

[66] Spiros Papadimitriou, Hiroyuki Kitagawa, Phillip B Gibbons, and Christos Faloutsos. Loci: Fast Outlier Detection Using the Local Correlation Integral. In *Data Engineering, 2003. Proceedings. 19th International Conference on*, pages 315–326. IEEE, 2003.

[67] Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. LoOP: Local Outlier Probabilities. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 1649–1652. ACM, 2009.

[68] Markus Goldstein and Andreas Dengel. Histogram-based Outlier Score (HBOS): A fast Unsupervised Anomaly Detection Algorithm. 2012.

[69] W3Schools. Onload Event. `http://www.w3schools.com/jsref/event_onload.asp`, August 2014.

[70] Mozilla Developer Network (MDN). Window.onblur. `https://developer.mozilla.org/en-US/docs/Web/API/Window.onblur`, August 2014.

[71] Mozilla Developer Network (MDN). Window.onfocus. `https://developer.mozilla.org/en-US/docs/Web/API/Window.onfocus`, August 2014.

[72] Mozilla Developer Network (MDN). Building an Extension. `https://developer.mozilla.org/en-US/docs/Building_an_Extension`, August 2014.

[73] Mozilla Developer Network (MDN). Progress Listeners. `https://developer.mozilla.org/en-US/Add-ons/Code_snippets/Progress_Listeners`, August 2014.

[74] Mozilla Developer Network (MDN). Notification Box. `https://developer.mozilla.org/en-US/docs/Mozilla/Tech/XUL/notificationbox`, August 2014.

[75] All free download. Free Website Templates. `http://all-free-download.com/free-website-templates/`, July 2014.

[76] Mozilla Foundation. Measuring Add-on Startup Performance. `https://developer.mozilla.org/en-US/docs/Mozilla/Performance/Measuring_add-on_startup_performance`, July 2014.

[77] David J Sheskin. *Handbook of parametric and nonparametric statistical procedures.* crc Press, 2003.

[78] Mozilla Firefox. iMacros for FireFox. `https://addons.mozilla.org/en-US/firefox/addon/imacros-for-firefox/`, June 2014.

[79] Firebug. Firebug and Network Monitoring. `http://getfirebug.com/network/`, July 2014.

[80] Eleazar Eskin, Andrew Arnold, Michael Prerau, Leonid Portnoy, and Sal Stolfo. A Geometric Framework for Unsupervised Anomaly Detection. In *Applications of data mining in computer security*, pages 77–101. Springer, 2002.

[81] Jiong Zhang and Mohammad Zulkernine. Anomaly Based Network Intrusion Detection with Unsupervised Outlier Detection. In *Communications, 2006. ICC'06. IEEE International Conference on*, volume 5, pages 2388–2393. IEEE, 2006.

[82] Mozilla Foundation. Performance Best Practices in Extensions. `lhttps://developer.mozilla.org/en-US/Add-ons/Performance_best_practices_in_extensions`, July 2014.

[83] Mozilla Wiki. Buildbot/Talos. `https://wiki.mozilla.org/Buildbot/Talos#Talos`, July 2014.

[84] Vladan Djeric's blog. A Quick Firefox Startup Update. `https://blog.mozilla.org/vdjeric/2013/06/25/a-quick-firefox-startup-update/`, July 2014.

[85] Peter H Westfall. *Resampling-Based Multiple Testing: Examples and Methods for p-value Adjustment*, volume 279. John Wiley & Sons, 1993.

[86] Software is hard. Firebug Net Panel Timings. `http://www.softwareishard.com/blog/firebug/firebug-net-panel-timings/`, July 2014.

[87] Kevin Leary. Testing Page Load Speed with Firebug. `http://www.kevinleary.net/testing-page-load-speed-with-firebug/`, July 2014.