TOWARDS SECURE WEB BROWSING ON MOBILE DEVICES

A Thesis
Presented to
The Academic Faculty

by

Chaitrali Vijay Amrutkar

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in
Computer Science

College of Computing
Georgia Institute of Technology
May 2014

TOWARDS SECURE WEB BROWSING ON MOBILE DEVICES

Approved by:

Professor Patrick Traynor, Advisor
College of Computing
Georgia Institute of Technology

Professor Wenke Lee
College of Computing
Georgia Institute of Technology

Professor Mustaque Ahamad
College of Computing
Georgia Institute of Technology

Dr. Shobha Venkataraman
AT&T Labs – Research

Professor Nick Feamster
College of Computing
Georgia Institute of Technology

Date Approved: December 3rd, 2013

To my parents,

Ujwala and Vijay Amritkar,

for giving their children wings to fly

ACKNOWLEDGEMENTS

The PhD journey is long, strenuous and presents several hurdles along the way. For a bird to safely reach her destination, her flying skills need to be honed. Maintaining the right balance, avoiding obstacles and changing direction when necessary is crucial. I have been fortunate to have a mentor who opened and closed many doors for me at the right time and guided me through the difficult PhD process. I thank my advisor, Prof. Patrick Traynor for having faith in me and giving me the freedom to explore new territories. Prof. Mustaque Ahamad was the first to hire me at Georgia Tech as a graduate research assistant during my masters' days. For his guidance and continuing support through my PhD process, I thank him. More thanks to my committee members for their valuable feedback and insights: Prof. Wenke Lee for encouraging me to strive for the best, Dr. Shobha Venkataraman for her guidance during and after my summer internship at AT&T labs, and Prof. Nick Feamster for his constructive advice. Special thanks to my collaborators outside Georgia Tech, Prof. Paul van Oorschot and Kapil Singh whose guidance helped improve my work.

A flight is more fun if taken with others who want to reach similar heights as one's own. I had a great team in the CISEC lab with fellow students Italo, Saurabh, Young, Hank, David, Brad and Chaz. I will always remember the birthdays celebrated in the lab and hope the celebrations continue. I thank everyone for making the lab a fun place to work. I grew personally and professionally around everyone. Musheer, Santosh, Long, Brendan, Vijay, Bharat, and Terry made my days at GTISC memorable. I thank you all. I also thank Alfreda and Mary Claire for being kind, and taking care of my caffeine needs and travel reimbursements. Finally, thanks to Adam Allred for his energetic help in increasing CISEC's productivity.

Surrounding oneself with friends and laughter is crucial in a PhD. I made some very close friends at Georgia Tech who became my family away from home. My roommate Anushree is one of my closest friends today and has stood by me no matter what the situation. I thank her for all the dance lessons, south Indian food and shopping sprees. Joon, Srikanth, Sam, Samantha, Arpit; thank you all for the cookouts, movies and restaurant binges. More thanks to Denise, Catherine and Samantha for being great colleagues in organizing women@cc events. Rohit and Mihir for the fun times during masters, Tushar and Hrushikesh for the road trips, Shauvik for the technical talks, Gauri for making sure I eat, and John and Bertrand for the gala time at AT&T and sharing the PhD experience since. Finally, Amruta, Sharvari and Prachi, for being there for me for almost two deacades. I hope our childhood bonds never break.

Lastly and most importantly, the bird needed the confidence that she could fly and that she had the freedom to travel to the place of her choice. My family has been the most important support system through the PhD process. My parents' love and inspiration have been crucial during the lows as well as the highs. They have always encouraged us children to think on our own, make our own decisions and then take responsibility for the same, while ensuring us that they will catch us when we fall.

Going forward, I wish to be as hardworking as my dad and be able to garner the same respect and faith from my colleagues as he garners from his. His astonishing range of knowledge has always awed us around him. With my dad's busy schedule, mom held the fort at home. She juggled a hundred things at once, from looking after finances to us kids. In many difficult situations, her positivity has kept us all going forward. She is a strong lady and that inspires me. My brother Chaitanya is carrying the family tradition forward in the field of medicine. His discipline, clarity in thought, knowledge and the ability to remain calm under pressure has helped me many times through my PhD. Thank you for frequently visiting me in the USA and bringing a piece of home closer. Finally, my fiancè Pushkar has been a pillar of strength through

my PhD. His calm and humble nature, and the ability to inspire others with his own successes have had a positive impact on me. Discussing life, work and everything under the sun with him is exhilarating. I know that we have many many happy years ahead of us. I consider myself fortunate to be surrounded by a loving family and dedicate this thesis to them. My success is also theirs.

TABLE OF CONTENTS

DEDICATION . . . . . **iii**

ACKNOWLEDGEMENTS . . . . . **iv**

LIST OF TABLES . . . . . **xi**

LIST OF FIGURES . . . . . **xiv**

SUMMARY . . . . . **xviii**

I INTRODUCTION **1**

1.1 Thesis Statement 4

1.2 Contributions 4

1.3 Dissertation Outline 6

II RELATED WORK **8**

2.1 Web Browser Policies and Attacks 8

2.1.1 Access Control Policies 9

2.1.2 Attacks and Defenses 10

2.1.3 Browser Extensions 10

2.1.4 Browser Kernels and Operating Systems 11

2.2 Browser Security Indicators 12

2.2.1 Ineffective Security Indicators on Desktop Browsers 12

2.2.2 Improved Security Indicators 14

2.3 Malicious Webpages 14

2.3.1 DNS-based Approaches 14

2.3.2 Content-based and In-depth Inspection Techniques 16

2.4 The Mobile Web 17

2.4.1 Native Application Security 17

2.4.2 Mobile Web Security 18

III MEASURING SYSTEMIC WEAKNESSES IN MOBILE BROWSER SECURITY 21

3.1 Introduction 21

3.2 Overview 24

3.2.1 Methodology 24

3.2.2 Threat Model 25

3.3 User Event Routing 27

3.3.1 Experimental Evaluation 27

3.3.2 Attacks 29

3.3.3 Analysis 33

3.4 Boundary Control 33

3.4.1 Experimental Evaluation 34

3.4.2 Attacks 34

3.4.3 Analysis 36

3.5 Top Level Frame Navigation 37

3.5.1 Attack and Experimental Evaluation 37

3.5.2 Analysis 40

3.6 Discussion and Potential Solutions 41

3.7 Conclusion 45

3.8 Appendix 46

IV AN EMPIRICAL EVALUATION OF SSL INDICATORS IN MOBILE BROWSERS 49

4.1 Introduction 49

4.2 Background on the W3C Recommendations 51

4.2.1 Definitions 51

4.2.2 W3C Guidelines 56

4.3 Empirical Observations 58

4.3.1 Identity Signal: Availability 58

4.3.2 Certificates: Required Content 60

4.3.3 TLS Indicators . . . 61
4.3.4 Robustness: Visibility of Indicators . . . 68
4.3.5 Error Messages . . . 69
4.4 User Deception and Potential Attacks . . . 73
4.4.1 Deception Methods . . . 73
4.4.2 Attacks . . . 76
4.5 Additional results . . . 79
4.5.1 The Good . . . 79
4.5.2 The Bad . . . 81
4.5.3 The Silent . . . 81
4.6 Discussion and Concluding Remarks . . . 83
4.6.1 Discussion . . . 83
4.6.2 Conclusion . . . 85

V KAYO: DETECTING MOBILE MALICIOUS WEBPAGES IN REAL-TIME . . . 86

5.1 Introduction . . . 86
5.2 Motivation . . . 89
5.3 Methodology . . . 92
5.3.1 kAYO Feature Set . . . 93
5.3.2 Data Collection . . . 99
5.4 Implementation and Evaluation . . . 102
5.4.1 Model Selection and Implementation . . . 102
5.4.2 Evaluation . . . 103
5.5 Browser Extension . . . 112
5.6 Discussion . . . 114
5.6.1 Investigating False Positives . . . 114
5.6.2 Cross-channel threats . . . 115
5.6.3 Limitations and Future Work . . . 116
5.7 Conclusion . . . 117

VI FUTURE WORK **119**

6.1 Advancing Dialogue on Mobile Browser Security 119

6.2 Tools for Malicious Mobile Webpage Detection 120

6.3 Hybrid Mobile Applications 120

6.4 Unified Permission Systems for Mobile Web Apps 121

6.4.1 Background 124

6.4.2 Proposed Architecture 126

6.4.3 Discussion 133

VII CONCLUSION **139**

REFERENCES **140**

LIST OF TABLES

1 Details of the browsers used for experimental evaluation. We also evaluated Opera Mini 5.5.1, Android 2.2.1 and Android 2.3.3 on Nexus One and Android 4.0.1 on Galaxy Nexus. We observed the same vulnerabilities in both the old and new versions of Opera Mini and Android browsers (except Android 4.0.1). (*: The version numbers of these browsers were not apparent. We have used the default browsers shipped with the referenced version of the OS.) 24

2 Market Share of Popular Mobile Browsers as of April 2012 [14]. We cover approximately 90% of the mobile browsers in the market for our evaluation. 25

3 Summary of observed display-related vulnerabilities in candidate browsers and respective attacks possible (A ✓depicts that attack is possible). **1**) Equivalent vulnerabilities exist in mobile and tablet browsers with different rendering engines. **2**) Mobile, tablet and desktop browsers from the same vendor do not necessarily implement the same code to handle display elements in different settings. **3**) Desktop browsers are more compliant with security policies for display. 41

4 Gazelle's (the ideal column) policies for a landlord to access a cross-origin tenant's position, dimensions, pixels and URL location. R: Read access. W: Write access. **R***: Android mobile, Android tablet on Xoom, Nokia Mini-Map and Opera Mini browsers allow a landlord to read user interaction with its cross-origin tenant. This vulnerability breaches the access control policy for the tenant's 'pixels' allowing the landlord to launch the user interaction interception attack. 44

5 Gazelle's (the ideal column) policies for a tenant of a cross-origin landlord to access its own position and dimensions on the landlord's page, its URL location and its pixel content. R: Read access. W: Write access. **RW***: Android mobile, iPhone and iPad2 Safari, Opera Mini and Opera Mobile allow a cross-origin tenant to write self dimensions and are thus susceptible to the phishing and password stealing attacks. IE Mobile does not allow a tenant to read self dimensions. This may allow a malicious landlord to shrink the tenant's dimensions to 'zero' without any notice to the tenant. 45

6 Details of the browsers used for experimental evaluation. (*: The version numbers of these browsers were not apparent. We have used the default browsers shipped with the referenced version of the OS.) . 57

7 Results of experiments on candidate mobile browsers to test compliance with the first two W3C guidelines given in Section 4.2.2. Each guideline column consists of sub-columns stating the experiments performed on the browsers. An × implies that the browser does not comply with the respective W3C guideline. A · implies that the browser complies with the respective W3C guideline. 59

8 Results of experiments on traditional web browsers to test compliance with the same guidelines as given in Table 7, with × and · symbols also implying the same. Note that all the desktop browsers are compliant. 59

9 Results of experiments on candidate mobile browsers to test compliance with the W3C guidelines 3a and 3b given in Section 4.2.2. The symbol notation is as defined in Table 7. 's': Implies that the `https` URL prefix is present on the 's'econdary interface. 62

10 Results of experiments on traditional web browsers to test compliance with the same guidelines as Table 9. The symbol notation is as defined in Table 7. 'p': Implies that the `https` URL prefix is present on the 'p'rimary interface. 62

11 Results of experiments on candidate mobile browsers to test compliance with the W3C guidelines 3c and 4 given in Section 4.2.2. The symbol notation is as defined in Table 7. 's': Implies that the `https` URL prefix is present on the 's'econdary interface. 66

12 Results of experiments on traditional web browsers to test compliance with the same guidelines as Table 9. The symbol notation is as defined in Table 7. 'p': Implies that the `https` URL prefix is present on the 'p'rimary interface. 67

13 Results of experiments on traditional web browsers to test compliance with the W3C guidelines 5a, 5b and 5c given in Section 4.2.2. The symbol notation is as defined in Table 7. NA: Implies that the concerned experiment is not applicable to that browser, the reasoning can be found in the text. (*: Our view is that a browser should display a warning message for a webpage holding mixed content, to avoid misleading users trained to interpret SSL indicators to mean that the (entire) webpage is secured.) $\times^*$: Implies that the browser fails to warn a user according to our view. 70

14 Results of experiments on traditional web browsers to test compliance with the same guidelines as Table 11. The symbol notation is as defined in Table 7 and Table 11. 70

15 Summary of potential attacks on candidate mobile browsers. A × implies that the attack is possible. A · implies that the corresponding attack is not possible on the browser. 73

16 Results of the support for SSLv2, the null cipher, DES-CBC-SHA (weak cipher) and whether browsers differentiate between EV-SSL and SSL certified webpages. The symbol notation is as defined in Table 7. 80

17 The 44 features of kAYO from four categories. According to our knowledge, the category of mobile specific features is studied for the first time in this work. The significance of these features is described in Section 5.4.2. 93

18 Indicators of mobile specific webpages extracted by manual analysis of the top-level mobile and desktop webpages of the 1,000 most popular websites on Alexa. We identified one top-level domain (TLD), nine subdomains and seven URL path prefixes. 99

19 Comparison of kAYO with Cantina, a technique using static webpage features to detect phishing webpages in real-time. kAYO's evaluation set size is over two orders of magnitude larger than that of Cantina. Moreover, kAYO's feature extraction process is two orders of magnitude faster than Cantina. Cantina's functionality is dependent on external tools unlike kAYO and Cantina works well only on webpages written in the English language. kAYO does not have these drawbacks. 105

20 Comparison of kAYO with five existing static analysis techniques that detect malicious desktop webpages. kAYO provides the lowest false positive rate on an evaluation set twice as large as the one used by other techniques. kAYO also considers mobile web threats, whereas, the other techniques are focused on detecting desktop web threats. . 106

21 Comparison of kAYO with five existing static analysis techniques that detect malicious desktop webpages. kAYO's feature extraction process is 10 times faster than the fastest existing technique [188] and classification time is 100 times faster than the fastest existing technique [153]. kAYO is the *only* technique that considers mobile specific features of webpages. 107

LIST OF FIGURES

1 **Left image**: Fake image advertisement of sales in San Francisco on the `www.landlordattacker.com` website; **Right image**: The mesothelioma ad from Google AdSense placed directly below the enticing fake sales ad image by malicious landlord. A user clicking on the mesothelioma ads [1] earns the landlord attacker more money. The landlord places the honest mesothelioma ads from AdSense in an iframe and overlays it with the more enticing images of sales in San Francisco to increase the rate of clicks. When a user clicks on the fake sale ad in San Francisco, the mesothelioma ad is clicked benefiting the landlord attacker. The Opera Mini (pictured), Android mobile, Android tablet on Xoom and Nokia Mini-Map browsers are vulnerable to the click fraud attack. 28

2 Login CSRF attack on Yahoo's sign in page. **Left image**: Image overlapping the `www.yahoo.com` iframe on `www.landlordattacker.com`. The text areas for entering 'solution' of the CAPTCHAs are placed exactly over the email and password fields on `yahoo.com`. The verify button is placed exactly above the 'sign in' button of `yahoo.com`. The two CAPTCHAs are the real email and password of the attacker's Yahoo account.; **Right image**: Login page of `www.yahoo.com` included in an iframe on `www.landlordattacker.com`, placed below the image. The Android mobile (pictured), Android tablet on Xoom, Opera Mini and Nokia Mini-Map browsers are vulnerable to this attack. 31

3 **Left image**: Layout of the malicious and honest widgets on the mashup webpage. 'ATTACKER' is a malicious widget and Amazon and YouTube are honest widgets; **Right image**: The browser allows a cross-origin tenant to write its own dimensions. The malicious widget expands its own dimensions and masquerades as the honest Amazon and YouTube widgets on the browser. It pushes the honest widgets south and launches a phishing attack on the user. This attack works in the iPhone Safari (pictured), Android mobile, iPad2 Safari, Opera Mini and Opera Mobile browsers. 35

4 **Left image**: `www.aol.com` webpage containing a cross-origin malicious advertisement. The browser displays only the 'title' of the page and does not display the address bar.; **Right image**: Due to the top-level frame navigation policy, the malicious ad can redirect the top-level window to `www.attacker.com`, which looks exactly the same as AOL's website, thereby launching a phishing attack. The user cannot detect the attack since the address bar containing the URL of the top window is not included in the mobile browser's view due to space constraint. The Nokia Mini-Map and Blackberry Mango browsers are the most susceptible to this attack. However, all other mobile and tablet browsers (except Chrome Beta and iPad2 Safari) are also susceptible to this attack due to address bar not being persistently available while browsing. 38

5 Identity information displayed by Firefox Mobile. 60

6 Blackberry Mango browser rendering a mixed content webpage. Note that the webpage contains a Google map obtained over an http connection. Although the webpage holds mixed content, the browser displays the padlock icon as well as the https URL prefix indicators. This behavior fails to meet with guideline 3a. 63

7 Security indicators on the primary interface (address bar) of all the mobile and tablet browsers. Every browser has three screenshots of the address bar: from top to bottom, the websites are Google over an `http` connection, Gmail over a secure connection with an SSL certificate and Bank of America over a secure connection with an EV-SSL certificate. 64

8 The address bar of the Android browser when a webpage over SSL is loaded. The browser places the favicon adjacent to the lock icon, thereby violating the W3C guideline 3b described in Section 4.2.2. The star icon to the right of the address bar is to bookmark the webpage. 65

9 Danger message on iPhone Safari when a website presenting a self-signed certificate is accessed. This message interrupts the user and also inhibits the user from proceeding without interacting with the danger message first. Note that the website's URL has been anonymized for submission. 72

10 Normalized density curves of static features when measured on mobile and desktop webpages. There is substantial difference between the distributions of the number of (a) iframes, (b) Javascript and (c) redirections when measured on mobile and desktop versions of the same websites, whereas, the distribution of the number of (d) IP addresses is similar. 89

11 Number of mobile specific websites found in every 10,000 websites in the top 1,000,000 URLs on Alexa. 100

12 The final ROC curve for kAYO's logistic regression model with regularization. 103

13 *Ex1*: Results of a model trained on desktop webpages using desktop features studied in earlier techniques and then tested on mobile webpages. *Ex2*: Results of a model trained on mobile webpages by adding mobile specific features to the feature set and tested on mobile webpages. Ex1 shows that a model trained on desktop pages using static features from earlier desktop-specific techniques, when applied to mobile webpages performs poorly. However, when a model is trained with the same static features and additional mobile specific features exclusively on a mobile datatset, the results of testing on a mobile dataset improve significantly as seen in Ex2. 108

14 The Pearson Coefficient Correlation (PCC) of each of the features extracted in kAYO with the label (malicious/benign), found using our evaluation dataset. Each point corresponds to the correlation of a feature with the label. In total there are 44 points corresponding to the 44 features of kAYO, including the newly identified features and the ones adopted from existing techniques. The Y value of each point depicts the predictive power of the corresponding feature i.e. PCC. The greater the absolute value of the PCC of a feature, the better predictive power of the feature. Note that all the PCC values are non-zero implying that every feature in kAYO's feature set is significant and impacts the result of classification. 110

15 Architecture of the mobile browser extension based on kAYO. User enters the URL he wants to visit in the extension toolbar and receives a response in real-time from our backend server about the maliciousness of the URL. If the URL is benign according to kAYO, the page of interest is rendered in the browser. Otherwise, the user is shown a warning message to not visit the URL. 111

16 (a): Chrome desktop browser informing the user of a potentially malicious webpage. The webpage is a known mobile phishing webpage. (b): The same webpage when rendered on the Chrome mobile browser, whose users are the real targets, does not provide any warning. (c): kAYO extension running on the Firefox mobile browser detects the webpage as malicious and warns the user. (d): Screenshot of command line processing at the extension server. 118

17 User interface for permission management of mobile web apps. **Left:** When a user clicks on the lock icon, the browser shows this interface to interact with permissions and certificates. **Right:** The browser provides an interface to view the status of the permissions requested by `www.foo.com` (domain in the address bar). The user has not stored hardware controls permissions, whereas he has stored the location, cookies and Internet access permissions. The cookie and Internet permissions are normal permissions granted without user consent. The location and hardware control permissions require explicit user consent. The user can easily revoke a permission by unchecking the box next to it. 130

SUMMARY

The Web is increasingly being accessed by portable, multi-touch wireless devices. Despite the popularity of platform-specific (native) mobile apps, a recent study of smartphone usage shows that more people (81%) browse the Web than use native apps (68%) on their phone [79]. Moreover, many popular native apps such as BBC depend on browser-like components (e.g., Webview) for their functionality [48]. The popularity and prevalence of web browsers on modern mobile phones warrants characterizing existing and emerging threats to mobile web browsing, and building solutions for the same. Although a range of studies have focused on the security of native apps on mobile devices, efforts in characterizing the security of web transactions originating at mobile browsers are limited.

This dissertation presents three main contributions: First, we show that porting browsers to mobile platforms leads to new vulnerabilities previously not observed in desktop browsers. The solutions to these vulnerabilities require careful balancing between usability and security and might not always be equivalent to those in desktop browsers. Second, we empirically demonstrate that the combination of reduced screen space and an independent selection of security indicators not only make it difficult for experts to determine the security standing of mobile browsers, but actually make mobile browsing more dangerous for average users as they provide a false sense of security. Finally, we experimentally demonstrate the need for mobile specific techniques to detect malicious webpages. We then design and implement kAYO, the first mobile specific static tool to detect malicious webpages in real-time.

CHAPTER I

INTRODUCTION

Internet connected mobile devices are going to outnumber humans in the year 2013 [88, 152]. Moreover, global mobile data traffic is expected to increase 13-fold between 2012 and 2017 [49]. Both platform-specific applications ("native apps") and browser-based applications ("web apps") enable mobile device users to perform security sensitive operations such as online purchases, bank transactions and accessing social networks. The distinction between native apps and web apps on mobile devices is increasingly being blurred. Many popular native apps, such as BBC, depend on browser-like components (e.g., Webview) for their functionality [48]. Moreover, as HTML5 becomes universally deployed and mobile web apps directly take advantage of device features such as the camera, microphone and geolocation, the difference between native and web apps will vanish almost entirely. A recent study of smartphone usage [79] shows that more people (81%) browse the Web than use native apps (68%) on their phone. Over 85% of handsets shipped globally in 2011 included some form of browser and it is expected that over 2.1 billion mobile devices will have a web browser component by 2016 [33]. This trend and the increasing use of web browsers on modern mobile phones warrant characterizing existing and emerging threats to mobile web browsing, and building solutions for the same. Although a range of studies have focused on the security of native apps on mobile devices, efforts in characterizing the security of web transactions originating at mobile browsers are limited.

Mobile web browsers have long underperformed their desktop counterparts. Whether by implementing limited alternative standards such as WAP [200] or incomplete versions of HTML, the first mobile browsers provided a meager set of capabilities and

attracted only a small number of early adopters. However, recent improvements in processing power and bandwidth have spurred significant changes in the ways users experience the mobile web. Modern mobile browsers provide rich functionality equivalent to their desktop counterparts using web technologies such as HTML, JavaScript, and CSS. Furthermore, browsers on mobile platforms now build on the same or similarly capable rendering engines used by many desktop browsers [40, 42]. Other features of mobile browsers include support for cryptographic tools including SSL/TLS and the corresponding user interfaces to convey SSL/TLS security implemented by websites to the end user. All these features have allowed mobile users to become increasingly reliant upon browsers to enable sensitive personal, social and financial exchanges.

Despite the apparent similitude between functionality offered by desktop and mobile browsers, the browsing experience on mobile devices is considerably different. This difference can be largely attributed to the dramatic reduction of screen size and the ability of invoking mobile specific functionality (e.g., SMS) through the web browser. These differences impact the design of web browsers and webpages built specifically for mobile devices, which in turn might lead to a number of security consequences. First, due to the limitations in the screen real estate, existing desktop browser software was not directly ported to mobile devices. Accordingly, while many mobile browsers bear the name of related desktop applications, their internal components might differ. The impact of these changes on security has not previously been evaluated. Second, in spite of the availability of SSL/TLS, mobile users are regularly becoming the target of malicious behavior. A 2011 report indicates that mobile users are three times more likely to access phishing websites than desktop users [80]. Security indicators (i.e., certificate information, lock icons, cipher selection, etc.) in web browsers offer one of the few defenses against such attacks. A user can view different security indicators and related certificate information presented by the browser to

offer signals or clues about the credibility of a website. Although mobile and tablet browsers appear to support similar security indicators when compared to desktop browsers, the reasons behind the increasing number of attacks on mobile browsers are not immediately clear. Finally, reduced screen size and availability of rich functionality also impacts the structure of webpages built specifically for mobile platforms. The content, functionality and layout of webpages have regularly been used to perform static analysis to determine maliciousness in the desktop space [84, 147, 176]. Features such as the frequency of iframes and the number of redirections have previously served as strong indicators of malicious intent. Due to the significant changes made to accommodate mobile devices, such assertions may no longer be true. For example, whereas such behavior would be flagged as suspicious in the desktop setting, many popular benign mobile webpages require multiple redirections before users gain access to content. Previous techniques also fail to consider mobile specific webpage elements such as calls to mobile APIs. For instance, links that spawn the phone's dialer (and the reputation of the number itself) can provide strong evidence of the intent of the page. New tools are therefore necessary to identify malicious pages in the mobile web.

To begin the effort of making mobile browsing secure, it is essential to understand the state-of-the-art of security in mobile browsers, and analyze the similarities between desktop and mobile browsers. This analysis can assist browser vendors with decisions of reusing security features from the desktop environment into the mobile environment to avoid duplication of effort. Browser vendors can also evade repeating already solved errors in desktop browsers in the corresponding mobile versions. Second, it is vital to understand the similarities and differences across the diverse browser software on popular mobile platforms. This evaluation can provide insight into the security impact of similar vulnerabilities in web browsers built by different vendors.

Furthermore, identifying similarities between different browsers can also facilitate formulating mobile specific standards for prevalent security problems. Third, studying the structural differences in mobile and desktop webpages will help build robust tools that consider the impact of changes in mobile webpages on security. Finally, stronger permission systems are necessary to manage the dynamic nature of mobile web apps and multiple access requests to sensitive information and hardware.

1.1 Thesis Statement

The goal of this thesis is to investigate the factors affecting security of the mobile web to improve the design and implementation of mechanisms for securing mobile web browsing. We argue that mobile web is different from the desktop web and thus demands independent evaluation and new techniques to protect sensitive information. Based on our evaluation of popular mobile browsers and mobile specific webpages, we propose the following thesis statement.

Mobile browsers, webpages and user interfaces significantly differ from those in the desktop environment thereby profoundly impacting security. Making the impact of limited display and mobile specific functionality integral to the design of web security solutions for mobile platforms identifies and addresses new threats.

1.2 Contributions

This dissertation makes the following contributions:

Perform the first comprehensive and systematic evaluation and comparison of security of desktop and mobile browsers: Modern mobile browsers now build on the same or similarly capable rendering engines used by many desktop browsers and also enable SSL/TLS transactions. We analyze SSL/TLS security indicators and display security on ten mobile (Android Mobile, Blackberry (Mango), Blackberry (Webkit), Chrome Beta, Firefox Mobile, Internet Explorer (IE) Mobile, Nokia Browser, Opera Mini, Opera Mobile and iPhone Safari) and three tablet (Android on Motorola Xoom,

Android on Samsung Galaxy and iPad2 Safari) browsers. We then compare the security standing of these mobile browsers with five most popular desktop (Chrome, Firefox, Internet Explorer, Opera and Safari) browsers. Our analysis covers over 90% of the mobile browser market and over 95% of the desktop browser market by download.

Identify new display security vulnerabilities in modern mobile browsers and implement real world attacks: We identify previously unknown erroneous display security policies in user event routing and boundary control, and implement multiple attacks that demonstrate their seriousness. Even though many mobile browsers rely on the same rendering engines as their desktop counterparts, our experiments demonstrate that mobile browsers are vulnerable to attacks not previously seen in the desktop space. Additionally, we exploit the conflict between usability and security in the mobile environment with limited screen estate to show that adopting some policies from desktop browsers exposes mobile browsers to new phishing attacks.

Demonstrate that the incomplete and inconsistent nature of SSL/TLS indicators in mobile browsers preclude experts from determining the security of web transactions: We experimentally illustrate that all popular mobile and tablet browsers fail to meet, in numerous instances, the recommendations in the W3C guidelines for user interface of security information, whereas in comparison desktop browsers largely follow the guidelines. We outline attacks on mobile browsers, such as phishing and undetectable man-in-the-middle, enabled by failure to properly follow these guidelines. Furthermore, we highlight missing security indicators, e.g., extended validation (EV) SSL indicators.

Design and implement the first mobile-specific static tool to detect malicious webpages in real-time: We demonstrate that mobile specific webpages differ significantly from their desktop counterparts in content, layout and functionality. We design and

implement kAYO, a fast and reliable mechanism that distinguishes between malicious and benign mobile webpages. kAYO makes this determination based on static features of a webpage ranging from the number of iframes to the presence of known fraudulent phone numbers. First, we experimentally demonstrate the need for mobile specific techniques and then identify a range of new content-based static features that highly correlate with mobile malicious webpages. We then apply kAYO to a dataset of over 350,000 known benign and malicious mobile webpages and demonstrate 90% accuracy in classification. Moreover, we discover, characterize and report a number of webpages missed by Google Safe Browsing and VirusTotal, but detected by kAYO. Finally, we build a browser extension using kAYO to protect users from malicious mobile websites in real-time.

Research impact: The newly discovered mobile browser vulnerabilities have been acknowledged and a subset of them addressed [2–4] by some browser vendors in the latest version of their browsers. The work on display security of mobile browsers (Chapter 3) was recognized as one of the top 10 papers of 2012 at the national level 'CSAW AT&T Best Applied Security Paper Award' competition. Moreover, it won the institute-level 'SAIC Best Student Paper Award' and the 'Best Demo' prize at the College of Computing research day at Georgia Tech. The second piece of this thesis (Chapter 4) was recognized as the 'Best Student Paper' at the Information Security conference 2012 and was covered by several media outlets [45,53,90,119,186,199]. The third and final piece of this thesis on detecting mobile malicious webpages (Chapter 5) has led to a patent.

1.3 Dissertation Outline

The goal of this dissertation is characterizing security of modern mobile browsers and implementing new mechanisms to secure web browsing on mobile devices. Chapter 3 provides details on the newly discovered display security vulnerabilities in modern

mobile browsers. We then discuss real-world attacks that exploit these vulnerabilities and also propose defenses. Additionally, we give an example of a universally adopted security policy that makes mobile browsers more vulnerable to phishing attacks than desktop browsers. Chapter 4 studies the impact of the small screen size of mobile devices on implementation of SSL/TLS indicators in browsers. We experimentally demonstrate that mobile browser vendors have implemented incomplete and inconsistent subsets of SSL/TLS indicators usually found in desktop browsers. We then discuss the impact of the unavailability of these indicators and outline potential phishing and man-in-the-middle attacks on security experts accessing mobile browsers. After studying security vulnerabilities in mobile browsers, we focus on mobile webpages in Chapter 5. We demonstrate the structural differences in desktop and mobile webpages through a series of experiments. We then characterize the consequence of these changes on existing static tools to detect desktop malicious webpages to show the need for mobile-specific tools. By selecting novel and existing static features of webpages relevant to mobile, we build the first technique to detect mobile malicious webpages in real-time.

Chapter 6 discusses our ongoing work on building new permission systems for mobile web apps. We study the impending changes in mobile web apps due to the introduction of HTML5 and web API suites such as Firefox Boot2Gecko [51]. We then provide a brief overview of our proposed architecture and future work. Chapter 7 offers concluding remarks.

CHAPTER II

RELATED WORK

Browsers and websites are the core components of web browsing. A web browser is a software application for retrieving, presenting and traversing information resources on the World Wide Web, whereas a webpage is information written in a document so that it renders correctly in a web browser. Securing each of these components individually is important to secure the end-to-end browsing experience. A web browser implements several security policies to protect users and individual websites from attacks [216]. While most of these policies are embedded in the browser's code, others are user facing (e.g., SSL/TLS indicators). Malicious browser extensions and plugins can also compromise private information of a user. Therefore, web browsers employ techniques to sandbox potentially malicious untrusted extensions [69] and plugins.

Simply securing the browser does not protect users from all web-based attacks. Attackers build malicious webpages to steal a user's identity or other sensitive information such as passwords [104] and credit card numbers [172]. Traditionally, browser-based attacks originated from bad websites. However, due to poor security policies of web applications or vulnerabilities in the software supporting websites [47], attackers have recently been successful in compromising large numbers of trusted web sites to deliver malicious payloads to unsuspecting visitors [141].

2.1 Web Browser Policies and Attacks

Design flaws in security policies, implementation errors, and trade-offs between performance and security lead to attacks on web browsers. Browsers implement several defense techniques against potential attacks, including access control policies for browser resources.

2.1.1 Access Control Policies

The Same Origin Policy (SOP) [67] is the most widely used access control policy in modern browsers. The SOP protects the content owned by a principal (domain or website, e.g., `www.example.com`) from being modified by an untrusted principal (e.g., `www.attacker.com`). The SOP defines each principal based on the corresponding browser resource, which include the Document Object Model (DOM), network, cookies, other persistent state and display [191]. For example, a principal for the DOM resource is defined as the tuple ⟨protocol, domain, port⟩; whereas for the cookie resource, a principal is labeled by ⟨domain, path⟩. This incoherency in labeling principals leads to replay attacks and privilege escalation [191].

Older techniques for inter-frame communication lead to breach of authentication and confidentiality. The fragment identifier messaging method provided confidentiality without authentication, whereas the postMessage method provided authentication, but breached confidentiality. Barth et al. [72] proposed stricter policies for fragment identifier messaging by adopting ideas from well-known network protocols and modified the postMessage API to allow the sender specify an intended recipient. These access control policies were primarily focused on isolating cross origin components of webpages. Jackson et al. [133] recognized that the security policy of browsers provides no isolation between documents from the same origin (scheme, host, and port), even if those documents have different security characteristics. This lack of isolation leads to origin contamination vulnerabilities in a number of browser security features, such as cookies, encryption, and code signing. Other weaknesses in access control mechanisms such as frame navigation policies [72, 209], client-side browser state [134], cookie path protection, and display protection [209] also expose browsers and web applications to a range of attacks.

2.1.2 Attacks and Defenses

SQL injection [168] is one of the most prevalent security risks as of 2013 [169]. This attack is carried out by inserting malicious SQL statements into an entry field for execution. Another way of injecting client-side scripts into webpages viewed by other users is Cross Site Scripting (XSS) [68]. The persistent and non-persistent types of XSS, together have surpassed buffer overflows to become the most common publicly reported security vulnerability in recent years [197]. Other steadily rising browser threats include Cross Site Request Forgery (XSRF) [71], clickjacking [6, 7, 182] and phishing. Implementation errors in the browser code [55,74], slow adoption of security techniques [221] and incorrect handling of privileges in browser extensions [69] further increase the threats to the browser and the user.

A range of defenses have been proposed to protect browsers from attacks. To defend against login CSRF, Barth et al. [71] proposed that browsers implement the Origin header, which provides the security benefits of the Referer header while responding to privacy concerns that have lead to the widespread suppression of the Referer header. Another client-side defense that mitigates cross-origin CSS attacks was proposed by Huang et al. [129]. The authors advocate enforcing content type checking for style sheets loaded from cross-origins, even if the requesting page is in quirks mode. Other defense techniques against web attacks include enforcing new security policies [129] and algorithms [54, 68] in browsers, running tools for detecting JavaScript-based attacks [92, 120, 139, 177], and implementing security vulnerabilities scanners [65].

2.1.3 Browser Extensions

Malicious extensions exploit browser vulnerabilities to run their code with all the privileges and features as that supported by any native programming languages. Researchers have investigated vulnerabilities in extension platforms of Firefox [65, 137,

149] and most recently Chrome [85, 148].

One of the fundamental defenses against malicious browser extensions is privilege separation [184]. Similar to OpenSSH [175] and qmail [75], this concept has been applied to build several tools and frameworks for modern web apps [78, 110, 142, 143, 161]. Moreover, studies have established that privilege separation has value in software projects that employ security experts (e.g., browsers [101]). Yet another protection technique from malicious plugins is implementing policies for document access, persistent state, network connections and other devices [122]. The plugin is required to run in a separate process from the browser and all interactions with the underlying system are performed through the browser. Finally, permission systems for browser extensions are popular in defending against malicious extensions [69,111,124].

2.1.4 Browser Kernels and Operating Systems

Websites include a number of cross-domain elements for rich features and user experience. Therefore, it is important to provide strong isolation between cross-domain principals in a browser to ensure code integrity and confidentiality. The OP Web browser [123] was the first to design a small browser kernel to enforce new browser security features and handle resources. The authors broke the web browser into several distinct and isolated components based on processes and made all interactions between these components explicit. The OP browser kernel then managed each of the components and interposed on communications between them. The OP browser allows any security model to be specified with their framework. However, this flexibility comes with a cost. The OP browser requires intimate interactions between browser components, such as JavaScript interpreter and HTML engine to use IPC and be inspected by the browser kernel. When targeting a specific security model, such as that of existing browsers, this additional IPC cost does not add any benefits

since isolating browser components within an instance of a webpage provides no additional security protection. Furthermore, the OP browser kernel does not provide cross-principal display protection. The Gazelle [209] browser provides cross-principal display protection and also reduces the cost of separating browser components within the same instance of a webpage by removing the requirement of IPC communication.

The security architecture of the Google Chrome browser [73] also repudiates the monolithic browser architecture that combines the "user" and the "web" into a single protection domain. Chromium has two modules than run in separate protection domains: a browser kernel, which interacts with the underlying operating system, and a rendering engine, which runs with restricted privileges in a sandbox. All these secure web browsers are built on top of commodity operating systems and include complex user-mode libraries and shared system services within their trusted computing base (TCB). The Illinois operating system and browser [198] reduce the TCB for web browsers drastically and simplify browser-based systems. The authors expose browser-level abstractions at the lowest software layer to remove almost all traditional OS components and services from the TCB by mapping browser abstractions to hardware abstractions directly.

2.2 Browser Security Indicators

Traditional desktop browsers contain user facing security indicators in addition to the security techniques embedded in the browser code. A range of security indicators are displayed in the chrome of the browser including the lock icon, the `https` URL prefix, and public key certificates.

2.2.1 Ineffective Security Indicators on Desktop Browsers

Each website provides its certificate information to the browser and the browser in turn conveys the same to the user using graphical and textual indicators. Certificates and other SSL/TLS indicators are meant to provide simple cues to the user about the

identity of the website and protection from eavesdroppers. However, several studies have indicated that these security cues used in desktop browsers go unnoticed [97,100, 185,192,211] or are absent in websites [193]. In a study conducted by Dhamija et al., desktop web browser users were challenged to identify phishing attacks in the presence of phishing and fraudulent certificate warnings [97]. 23% of their subjects completely ignored the passive or non-interruptive phishing warnings, and 68% of subjects quickly clicked through the active or interruptive fraudulent certificate dialogs. Another study by Akhawe et al. [135] used Mozilla Firefox and Google Chrome's in-browser telemetry to observe 25 million warning impressions *in situ*. The authors found that users continued through a tenth of Mozilla Firefox's malware and phishing warnings, a quarter of Google Chrome's malware and phishing warnings, and a third of Mozilla Firefox's SSL warnings. Moreover, it was observed that users rarely click on the explanatory links such as "More Information" or "Learn More".

Although domain name mismatches between certificates and websites are observed often [206], Sunshine et al. [196] showed that users ignore TLS warnings for domain name mismatches, and showed that users ignore TLS warnings for expired certificates and unknown CAs. Moreover, a majority do not understand these warnings. The lock icon is the security indicator most often noticed [100, 211]. However, even when used as a security cue by users, many do not fully understand its meaning [97,98,100] and its absence also often goes unnoticed [97]. Additionally, the majority of users who rely on the lock icon remain unaware of its identity feature [97,100,115,211] and do not reliably understand the concept of certificates [97, 98]. Indicators for newer technologies such as EV-SSL have also been shown to be ineffective to convey better security to the user as compared to a simple SSL certificate [76, 136].

2.2.2 Improved Security Indicators

Several techniques have been proposed to design better security indicators to prevent potential attacks such as phishing. Researchers have proposed better warnings [196], more effective interface dialogues [76], browser plugins [91], trusted path from the browser to the human user [214] and mandatory security indicators [126] to help users make correct security decisions. Other proposed security mechanisms include disabling JavaScript in the user browser and forcing persistent visibility of the browser's location line [113]. Dynamic Security Skins [98] allow a remote web server to prove its identity in a way that is easy for a human user to verify and hard for an attacker to spoof. Finally, efforts have been taken [29,34–36,41] to standardize security indicators and thus minimize confusion across browsers.

2.3 Malicious Webpages

Simply securing the web browser alone cannot protect a user from all web-based attacks. Attackers build malicious webpages to steal a user's identity or other sensitive information such as passwords or credit card numbers.

2.3.1 DNS-based Approaches

A popular approach in detecting such malicious activity on the web is by leveraging distinguishing features between malicious and benign DNS usage. The first study [205] in this direction proposed to collect real-world DNS data for analyzing malicious behavior. The results of the passive DNS analysis showed that malicious domains that are used in fast-flux networks exhibit behavior that is different than benign domains [217]. Antonakakis et al. [63] added to the passive monitoring idea by proposing Notos, a detection scheme that dynamically assigns reputation scores to domain names whose maliciousness is yet to be discovered. The premise behind Notos is that agile malicious uses of DNS have unique characteristics and thus malicious

use of DNS can be distinguished from benign use. To this end, the authors analyze a number of features from three categories, network-based features, zone-based features and evidence based features. Notos is unable to detect malicious domains that are mapped to a new address space each time and never used for other malicious purposes again. This limitation is addressed by yet another passive DNS monitoring system called EXPOSURE. EXPOSURE uses time-based features which account for short-lived domains. Other efforts to identify malicious domains include more passive DNS monitoring tools [173, 217] and active DNS probing methods [127, 131]. Active DNS probing methods repeatedly query the domains that are advertised to be malicious by various sources (e.g., spam mails) to detect the abnormal behavior. The main drawback of active DNS analysis is the possibility of being detected by the miscreants who manage the domains under analysis. Passive DNS analysis, in comparison, is more stealthy because of its non-intrusiveness characteristics.

These techniques did not detect all types of web-based attacks. While some of these existing efforts focused solely on detecting fast-flux service networks [127, 164, 171, 210], another [77] can also detect domains implementing phishing and drive-by-downloads. Fast-flux service networks [127] are malicious systems that abuse Round-Robin DNS. Most of the efforts in detecting fast-flux service networks [127, 140, 164, 171] differ from each other only in the number of features used and the details of the classification algorithms. They are also limited to mainly studying fast-flux domains advertised through email spam. In particular, potential fast-flux domain names are extracted from the URLs found in the body of spam emails in a dataset. Then an active probing strategy is applied, which repeatedly issues DNS queries to collect information about the set of resolved IP addresses to classify each domain name into either fast-flux or non-fast-flux. Perdisci et al. [173] overcame the limitations of such techniques by performing passive analysis of recursive DNS (RDNS) traffic traces. A major drawback of these DNS based mechanisms is that they do not necessarily

provide deeper understanding of the specific malicious activity implemented by a webpage or domain.

2.3.2 Content-based and In-depth Inspection Techniques

Dynamic approaches using virtual machines [159, 176] and honeyclient systems [130, 157,163] provide deeper visibility into the behavior of a webpage. Honeyclient systems fully execute the contents of a webpage. This includes fetching the webpage, all the resources that are linked from it, and then interpreting the associated dynamic content, such as JavaScript code. The complete visibility into each webpage leads to a very low false positive rate and great accuracy. However, downloading and executing each webpage also impacts performance and hinders scalability of dynamic approaches. Each webpage can take anywhere from a few seconds to several minutes, depending on the complexity of the analyzed page.

This performance penalty can be avoided by using static approaches. The oldest static approach is signature-based techniques based on string patterns in malicious code, commonly used in anti-virus tools [32]. Such techniques can be easily evaded using obfuscation, thus suffering from high false negative rates [92,179]. These high false negative rates can be reduced by using static approaches that rely on the structural and lexical properties of a webpage and do not execute the content of the webpage. One such technique of detecting malicious pages is using statistical methods for URL classification based on a URL's lexical and host-based properties [114, 117, 144, 153]. Garera et al. used URL statistical techniques to classify phishing URLs [117]. A larger scale classification was carried out by Ma et al. [153] using lexical properties of URLs and registration, hosting, and geographical information of the corresponding hosts. All URL-based techniques usually suffer from high false positive rates. Using HTML and JavaScript features extracted from a webpage in addition to URL classification helps address this drawback and provides better results [84, 156, 212, 218]. Commonly

used features include visibility and size of iframe tags, and the number of script tags referencing external resources. Static approaches avoid performance penalty of dynamic approaches. Additionally, using fast and reliable static approaches to detect benign webpages can avoid expensive in-depth analysis of all webpages.

2.4 The Mobile Web

Mobile Internet users are growing rapidly [166]. Based on the current rate of change and adoption, mobile web usage will be greater than desktop Internet use by 2015. Mobile users access Internet using both native applications and web browsers. Despite 81% of mobile users browsing web on their phones using a web browser [79], the majority of security research in the recent years has focused on securing native applications on mobile devices.

2.4.1 Native Application Security

The Android, iPhone, Symbian and Windows operating systems use different types of permissions. The Android OS and iOS have been the most popular among researchers due to the popularity of the iOS platform and both widespread use and open source nature of the Android platform.

Detecting overprivilege in Android applications and studying its impact on users' private data has been a popular area of study. Enck et al. applied Fortify's Java static analysis tool to decompiled applications to study the applications' use of a small number of permissions and API calls [106]. Their analysis uncovered pervasive use or misuse of personal and phone identifiers, and deep penetration of advertising and analytics networks. Felt et al. detected overprivilege by manual classification of a small set of Android applications [111], whereas the Kirin [107] system used static analysis focusing on permissions and other application configuration data. Kirin relies solely on developer permission requests, rather than examining whether or how permissions are used by applications. Another permission overprivilege study examined

1,100 Android applications' permission requirements and used self-organizing maps to visualize which permissions are used in applications with similar characteristics [66]. Other tools to detect overprivilege include application source code analyzer [204], application package attribute analyzer [87], NLP to detect reasoning behind permission requests [170] and static analysis on Android APIs [118].

Several systems also studied the impact of application overprivilege on users' private data. TaintDroid [105] used system-wide dynamic taint tracking to identify privacy leaks in Android applications. By using static analysis, the authors studied a number of applications and confirmed the exfiltration of information. PiOS [102] performed static analysis on iOS applications for the iPhone. The PiOS study found that the majority of analyzed applications leaked the device ID and over half of the applications included advertisement and analytics libraries.

In addition to application overprivilege, host security is a growing concern on smartphones. OS-level protections such as Saint [167] and Security-by-Contract [96] provide enhanced security mechanisms for Android and Windows Mobile. These approaches prevent access to sensitive information; however, once information enters the application, no additional mediation occurs.Mulliner et al. [160] provide information tracking by labeling smartphone processes based on the interfaces they access, effectively limiting access to future interfaces based on acquired labels. Finally, Aquifer presents a policy framework and system for preventing accidental information disclosure in modern operating systems such as Android, iOS, and Windows 8 [162].

2.4.2 Mobile Web Security

Web browsers have become one of the most popular applications on today's smart phones. The mobile web research so far has focused on browser energy consumption analysis [201], device performance [93] and mobile latency [56]. However, there are limited or no efforts in securing web browsing on mobile devices. In addition to

malicious mobile applications affecting user privacy [102,105] and potentially harming the cellular network [202,203], the increasing user base of mobile platforms and mobile e-commerce have made mobile browsers an attractive target for attackers [24, 28, 31, 59, 95, 112, 165, 178, 183, 183]. Researchers have already begun to think about defending against attacks on mobile phones using smart CDNs [150]. Although mobile browsers will be targets of security attacks in the coming years, security issues in mobile browsers will be new since the devices have serious limitations compared to desktops. However, a large-scale security analysis of the differences between mobile and desktop browser software has not yet been performed.

In addition to the underlying code, the user interfaces of mobile browsers differ significantly from their desktop counterparts. The small display of mobile phones and tablet computers leads to adaptation in user facing security indicators in web browsers. Until now, almost all research efforts in the area of security indicators in browsers have been focused on desktop browsers. However, in light of the recent attacks targeted towards mobile browsers [28, 59] and considering how the mobile browser user interface differs from desktops, it is important to analyze and understand the security indicators used in mobile browsers. Although the W3C [35] guidelines consider mobile browsers in their definitions, a large-scale evaluation of the state-of-the-art security indicators in mobile browsers has not been carried out.

Finally, all the approaches for malicious webpage detection have focused on websites built for desktop browsers in the past. Although differences in mobile and desktop websites have been observed before [83], it is unclear how these differences impact security. Furthermore, the threats on mobile and desktop websites are somewhat different [112]. Static analysis techniques using features of desktop webpages have been primarily studied for drive-by-downloads on desktop websites [84, 176], whereas, the biggest threat on the mobile web at present is believed to be phishing [81]. Efforts

in mitigating phishing attacks on desktop websites include isolating browser applications of different trust level [116], email filtering [114], using content-based features [212, 218] and blacklists [151]. The best-known non-proprietary content-based approach to detect phishing webpages is Cantina [218]. Cantina suffers from performance problems due to the time lag involved in querying the Google search engine. Moreover, Cantina does not work well on webpages written in languages other than English. Finally, existing techniques do not account for new mobile threats such as known fraud phone numbers that attempt to trigger the dialer on the phone. Consequently, whether existing static analysis techniques to detect malicious desktop websites will work well on mobile websites is yet to be explored.

In summary, the mobile web is evolving rapidly. Most of the existing techniques in securing web browsing focus primarily on the desktop environment. However, due to the differences in the mobile and desktop environments, the threats in desktop might not translate directly to the mobile environment. Furthermore, the mobile web presents new threats due to the newly added functionality such as web APIs. Therefore, investigating security of mobile browsing independent of desktop browsing is crucial.

CHAPTER III

MEASURING SYSTEMIC WEAKNESSES IN MOBILE BROWSER SECURITY

3.1 Introduction

Mobile web browsers have long underperformed their desktop counterparts. Whether by implementing limited alternative standards such as WAP [200] or incomplete versions of HTML, the first mobile browsers provided a meager set of capabilities and attracted only a small number of early adopters. However, recent improvements in processing power and bandwidth have spurred significant changes in the ways users experience the mobile web.

Modern mobile browsers now build on the same or similarly capable rendering engines used by many desktop browsers [40,42]. Mobile browsers are so capable that, through APIs such as WebViews, many of the most popular mobile apps (e.g., BBC, Walgreens) [48,82] act as wrappers for the browser pointed to specific webpages. However, due to limitations in the screen real estate and memory, existing desktop browser software was not directly ported to mobile devices. Accordingly, while many mobile browsers bear the name of related desktop applications, their internal components might significantly differ. The impact of these changes on security has not previously been evaluated. Given the popularity of browsing on mobile devices [79,152], focusing on the security of mobile browsers is critical.

In this chapter, we perform the first large-scale security comparison between mobile and desktop browsers. While there are many potential areas for investigation, we focus on the issues of display security due to the screen constraints of mobile

devices. Given the often crowded layout of mobile webpages, we specifically investigate the behavior of overlapping HTML elements (and how browsers handle clicks - i.e., "user event routing"), behavior at the boundaries between non-overlapping items ("boundary control") and the impact of nonpersistent availability or complete absence of the address bar. We apply blackbox analysis across ten mobile, three tablet and five desktop browsers and demonstrate that many mobile and tablet browsers are vulnerable to new two classes of attacks due to inconsistent click-event routing and incorrect write policies. We illustrate that desktop browsers are not susceptible to these attacks and present solutions to address the new vulnerabilities. We then discover a third class of vulnerability resulting from a clash between considerations made for usability in mobile browsers and a universally implemented display policy, demonstrating that making usability considerations while creating mobile software is crucial and blind porting of traditional browser code to mobile devices can introduce unexpected vulnerabilities.

We make the following contributions:

- **Characterize display security disparity between the most popular mobile and desktop browsers:** We analyze display security on ten mobile (Android Mobile, Blackberry (Mango), Blackberry (Webkit), Chrome Beta, Firefox Mobile, Internet Explorer (IE) Mobile, Nokia Mini-Map, Opera Mini, Opera Mobile and iPhone Safari), three tablet (Android on Motorola Xoom, Android on Samsung Galaxy and iPad2 Safari) and five desktop (Chrome, Firefox, Internet Explorer, Opera and Safari) browsers. We use blackbox analysis as source code is not available for the majority of browsers. Table 3 on page 14 summarizes our findings.

- **Identify erroneous implementations of display security policies:** We

identify *previously unknown* erroneous policies in user event routing and boundary control and implement multiple attacks that demonstrate their seriousness. Even though many mobile browsers rely on the same rendering engines as their desktop counterparts, our experiments demonstrate that mobile browsers are vulnerable to attacks not previously seen in the desktop space.

- **Expose conflict between usability and display security:** We show that some re-implemented policies from desktop browsers, specifically Top-Level Frame Navigation [70], expose mobile devices to phishing when mobile browsers hide or completely eliminate indicators such as the address bar for reasons of usability. In particular, we demonstrate the ability to navigate users away from their intended destinations. *Our technique is new and does not use address bar spoofing similar to the phishing techniques studied earlier [112, 165].* We find that our technique enables a more dangerous and easy to launch attack, since it exploits a built-in policy in all web browsers instead of attempting to spoof the address bar in individual browsers.

Our analysis demonstrates that the discovered vulnerabilities are not isolated bugs; rather, *they are pervasive and affect all but one of the most popular mobile and tablet browsers in some capacity.*[1] We have communicated our results to various browser vendors who have acknowledged the presence of these vulnerabilities. Moreover, we argue that because an increasing number of apps rely on mobile browsers, that these issues are relevant to all mobile app developers. Our results are the first *comprehensive* study in display security and they provide strong evidence that the security of mobile browsers has taken steps backward when compared to desktop browsers.

[1]The Chrome Mobile browser was not susceptible to any of the attacks described in this work at the time of experiments (June 2011). However, the latest version of the Chrome Mobile browser (as of Dec 2013) minimizes the address bar on page rendering, thereby being susceptible to the attack described in Section 3.5.

Table 1: Details of the browsers used for experimental evaluation. We also evaluated Opera Mini 5.5.1, Android 2.2.1 and Android 2.3.3 on Nexus One and Android 4.0.1 on Galaxy Nexus. We observed the same vulnerabilities in both the old and new versions of Opera Mini and Android browsers (except Android 4.0.1). (*: The version numbers of these browsers were not apparent. We have used the default browsers shipped with the referenced version of the OS.)

Category	Browser Name	Version	Rendering Engine	Operating System	Device
Mobile	Android	2.3.6	Webkit	Android 2.3.6	Nexus One
	Blackberry	5.0.0	Mango	Blackberry OS 5.0.0.732	Bold 9650
	Blackberry	6.0.0	Webkit	Blackberry OS 6	Torch 9800
	Chrome Beta	0.16.4301.233	Webkit	Android 4.0	Galaxy Nexus
	Firefox Mobile	4 Beta 3	Gecko	Android 2.3.6	Nexus One
	Internet Explorer Mobile	*	Trident	Windows Phone 7.0.7004.0 OS	LG-C900
	Nokia Mini-Map	*	Webkit	Symbian S60	E71x
	Opera Mini	6.0.24556	Presto	Android 2.3.6	Nexus One
		5.0.019802	Presto	iOS 4.1 (8B117)	iPhone
	Opera Mobile	11.00	Presto	Android 2.3.6	Nexus One
	Safari	*	Webkit	iOS 4.1 (8B117)	iPhone
Tablet	Android	*	Webkit	Android 3.2.1	Motorola Xoom
	Android	*	Webkit	Android 3.1	Samsung Galaxy
	Safari	*	Webkit	iOS 4.3.5 (8L1)	iPad 2
Desktop	Chrome	15.0.874.106	Webkit	OS X 10.6.8	–
	Firefox	7.0.1	Gecko	OS X 10.6.8	–
	Internet Explorer	8.0.7600.16385	Trident	Windows 7	–
	Opera	11.52	Presto	OS X 10.6.8	–
	Safari	5.1.1	Webkit	OS X 10.6.8	–

3.2 Overview

This section discusses our experimental methodology and defines our threat model.

3.2.1 Methodology

We analyze the rendering differences between popular desktop and mobile browsers for security. The studied browsers are shown in Table 6. We have selected these browsers as they represent approximately 90% of mobile browsers in the market [14], as shown in Table 2.

We define a 'display element' as any HTML element that can color pixels on the screen. For example, `iframe`, `image`, `text`, `text area`, `link`, `table` and `button` all fall under display elements. However, HTML elements such as `head` or `option` do not qualify as display elements. We create customized scenarios to evaluate common interactions of cross-origin display elements: 1) when they overlap, 2) when they border each other and 3) when they are navigated to new sources. Given the tight

Table 2: Market Share of Popular Mobile Browsers as of April 2012 [14]. We cover approximately 90% of the mobile browsers in the market for our evaluation.

Browser Name	% **Market Share**
Opera	22.52
Android	21.18
iPhone Safari	19.85
Nokia	11.65
Blackberry	6.08
iPod Touch	3.72
Other (Firefox Mobile, IE on Windows Phone 7 OS etc.)	3.28

layout of many mobile webpages and the corresponding small screen sizes of the associated devices, characterizing such interactions is critical. We discover new classes of vulnerabilities in mobile browsers and evaluate their risk by implementing attacks exploiting the vulnerabilities. All the experiments were performed on browsers on real mobile phones, and are recreated in the respective emulators to create many of the figures throughout the chapter.

3.2.2 Threat Model

We consider two classes of adversaries. Each adversary attempts to attack other website principals and/or the user and exploit the constrained nature of a mobile device's display. Each adversary can identify the user's mobile browser and is knowledgeable of the display-related security vulnerabilities associated with that browser.

Landlord attacker: The *landlord attacker* is a malicious principal[2] who can host his own websites such as `landlordattacker.com`. For example, the owner of a phishing website such as `blankofamerica.com` imitating `bankofamerica.com` is classified as a landlord attacker. A 'tenant' is a principal who rents an area on a landlord's website to render his own content such as advertisements. After the landlord gets honest tenants on his website, he attempts to exploit the honest tenant and/or the honest user. The

[2]A principal is the owner of some web content. In general, one principal does not trust another with respect to his resources [208].

landlord cannot read or change parts of the content in the tenant's rented area on the screen (due to the Same Origin Policy[3]), but controls the external properties of the tenant's rented area. For example, the landlord can specify the dimensions, transparency and position of the tenant's area on his website. The landlord instead tries to attack the honest tenant and honest user by manipulating his own website display.

We note that not every user visiting the malicious website will be exploited. Depending on the vulnerability targeted by the landlord attacker, the honest tenant and honest user may be attacked only when `landlordattacker.com` is rendered in a vulnerable browser. Placing web advertisements, displaying popular content indexed by search engines and sending bulk e-mail to users are some of the techniques that the landlord attacker can use to attract users to his website [72].

Tenant attacker: The *tenant attacker* is a malicious principal who can rent an area of the display on a website owned by an honest landlord. For example, the tenant attacker can insert a malicious advertisement or widget into an honest website. Websites such as iGoogle allow any user having an account to upload a new widget. We assume that an honest user visits an honest website containing at least one tenant attacker area using a vulnerable mobile browser. The tenant attacker has knowledge of the display vulnerabilities in the popular mobile browsers. He manipulates the content of his rented area to attack the honest website and/or the user.

A successful exploit is able to:

1. Influence the state and logic of a victim website principal across Same Origin Policy boundaries, and/or

2. Deceive a user into performing unintended actions or sharing private data.

[3]The Same Origin Policy prevents a document or script loaded from one domain from getting or setting properties of a document from another domain [18, 181].

3.3 User Event Routing

Overlapping elements are common in many webpages. From drop-down menus to floating advertisements, the ability to overlay objects allows for content to be dynamically presented to the user. However, the interaction between such elements must be strictly defined, especially in cases when they are controlled by different origins. When two or more display elements share the same pixel on the screen, browsers must decide both a) which element can control the 'coloring' (display) of the pixel and b) which element owns and responds to the user access to that pixel (user event routing). For example, if a drop-down menu covers over an image and a user clicks in this shared screen area, the browser must decide whether the principal owning the image or the principal owning the menu will respond to a user's click action.

Although all browsers make these decisions, the security relevance of user event routing in overlapped elements has not previously been studied. Our evaluation demonstrates that while desktop browsers consistently route user actions to the topmost element, event routing is inconsistent across mobile and tablet browsers. *This inconsistency allows hidden elements to intercept user actions and potentially perform dangerous operations.* We first discuss the results of our evaluation of overlapped elements using the methodology in Section 3.2.1 and then present attacks exploiting the vulnerabilities.

3.3.1 Experimental Evaluation

Mobile and tablet browsers:

Inconsistent click-event reception: Click-event reception refers to a browser choosing the element that receives a user's click action in a stack of overlapped elements. In the Android mobile, Android tablet on Xoom, Nokia Mini-Map and Opera Mini browsers, a user's `onclick` event on an image is routed to the `onclick` events of buttons, text areas and links below the opaque image, thereby executing the events of the hidden

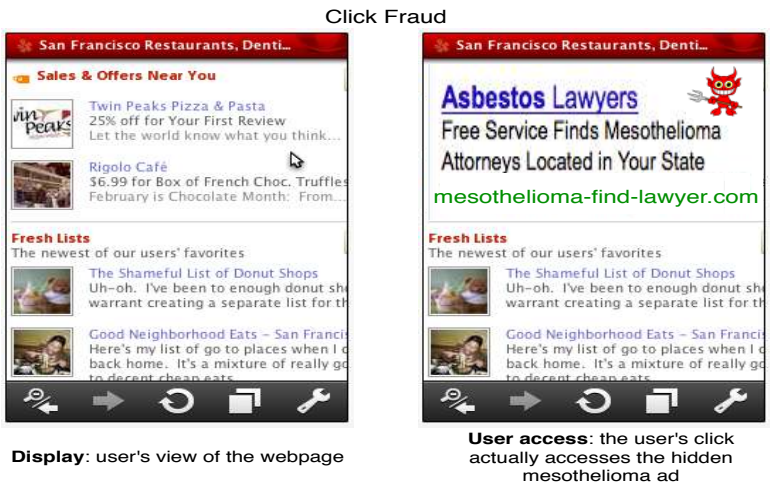

Figure 1: **Left image**: Fake image advertisement of sales in San Francisco on the `www.landlordattacker.com` website; **Right image**: The mesothelioma ad from Google AdSense placed directly below the enticing fake sales ad image by malicious landlord. A user clicking on the mesothelioma ads [1] earns the landlord attacker more money. The landlord places the honest mesothelioma ads from AdSense in an iframe and overlays it with the more enticing images of sales in San Francisco to increase the rate of clicks. When a user clicks on the fake sale ad in San Francisco, the mesothelioma ad is clicked benefiting the landlord attacker. The Opera Mini (pictured), Android mobile, Android tablet on Xoom and Nokia Mini-Map browsers are vulnerable to the click fraud attack.

elements. We note that only the events corresponding to the element directly situated below the area where a user clicks responds to the click action. Click events of all the elements situated below the image are not executed when the user clicks on the image.

In the Nokia Mini-Map and Opera Mini browsers, even if the top image has an `onclick` event associated with it, the `onclick` events of the buttons below the image are given preference. If the image on top does not have an event associated with it, the buttons below the image are clickable in the Android mobile and Android tablet on Xoom browsers.

Incorrect write policy: The Android mobile, Android tablet on Xoom, Nokia Mini-Map and Opera Mini browsers allow a user to write into the text areas in an iframe situated below an opaque image. When a user clicks on the portion of the image

overlapping any part of the text area below, the text area pops out on top and the user can write into the box.

Desktop browsers: The desktop browsers always route click and write events exclusively to the top element in a stack of overlapped elements.

3.3.2 Attacks

We present three novel techniques that exploit inconsistent click-event reception and incorrect write policies for overlapping elements.

1) Click Fraud: This attack is possible due to inconsistent click-event reception in overlapping elements. Click fraud occurs in pay-per-click advertising when a malicious principal creates illicit clicks on an ad by either tricking a real user or by imitating a legitimate user's click with a program. Such attacks generate revenue per click with no actual interest in the target of the ad's link. A popular pay-per-click advertising program is Google's AdSense. A malicious landlord or tenant website cannot manipulate the ad placed by Google (due to the Same Origin Policy) and thus cannot trick a legitimate user into clicking on an unwanted ad by disguising it with more enticing content.

Consider a malicious landlord principal who creates an AdSense account and embeds relevant content containing targeted keywords to attract high paying ads. The high paying ads [1] are generally not as popular as ads for discounts or coupons and thus are not clicked very often. A landlord attacker can carry out click fraud as shown in Figure 1, on a browser that allows a user to inadvertently access hidden content (links, buttons etc.) placed below an opaque element such as an image. The landlord attacker overlaps the mesothelioma ad (right) with more enticing and opaque content such as sales at local restaurants (left). If an honest user clicks the area containing

the attractive content from a vulnerable browser, the mesothelioma ad[4] below the attractive content will be clicked without the user's knowledge. Since the user's click is captured by the Google AdSense ad instead of the image on top, the malicious landlord illicitly benefits.

2) Login CSRF: This attack is possible due to inconsistent click-event reception and incorrect write policies. The intention of an attacker in a login Cross Site Request Forgery (CSRF) is to make the honest user's browser log in *as the attacker* into a legitimate website without any notice to the user. While seemingly counter-intuitive, such an attack allows an adversary to monitor operations executed by the user and steal their private information. For example, if an attacker successfully logs in into his Yahoo account from the victim's browser, the victim's actions on all of the websites (search, shopping, finance, health) belonging to Yahoo's single sign-on system will be recorded in the attacker's account. If the user makes a purchase at `shopping.yahoo.com` and enters his credit card details, the information will be stored in the attacker's profile. Note that the user will not be asked to sign-in since the attacker has already signed in in the user's browser. Previous work has leveraged a browser's network connectivity and a browser's state to launch a login CSRF attack [71]. We present a new mechanism to launch the login CSRF attack by exploiting the vulnerability of incorrectly handling user access to overlapped display elements in mobile browsers. Our method is more robust and not easy to detect since it exploits an in-built vulnerability in the browsers.

Consider a malicious website `landlordattacker.com`. The landlord includes a legitimate iframe containing the 'sign in' page of `www.yahoo.com` as shown in Figure 2 (right). The landlord then overlaps the iframe completely with an opaque image as shown in Figure 2 (left). The image shows enticing free content on the landlord's

[4]Mesothelioma is a cancer caused by inhaling asbestos and an ad costs $65.21 per click [15].

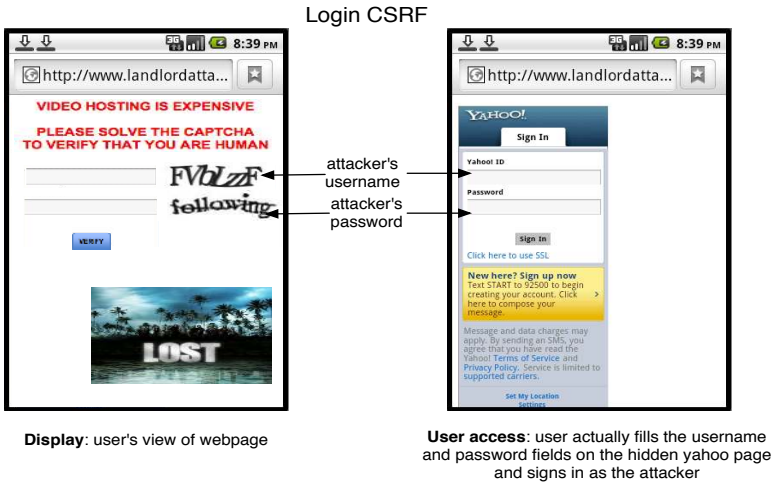

Figure 2: Login CSRF attack on Yahoo's sign in page. **Left image**: Image overlapping the `www.yahoo.com` iframe on `www.landlordattacker.com`. The text areas for entering 'solution' of the CAPTCHAs are placed exactly over the email and password fields on `yahoo.com`. The verify button is placed exactly above the 'sign in' button of `yahoo.com`. The two CAPTCHAs are the real email and password of the attacker's Yahoo account.; **Right image**: Login page of `www.yahoo.com` included in an iframe on `www.landlordattacker.com`, placed below the image. The Android mobile (pictured), Android tablet on Xoom, Opera Mini and Nokia Mini-Map browsers are vulnerable to this attack.

website and includes two image CAPTCHAs expected to be solved by the user to access the free content. The intention of the landlord attacker is to make the user enter the attacker's credentials into the hidden iframe below the opaque image. The landlord accomplishes this by setting the two CAPTCHAs to the email and password of the attacker's Yahoo account. For example, in Figure 2, *FVbLzzF* and *following* are the username and password respectively of the attacker's Yahoo account. The landlord attacker then carefully places each of the solution boxes of the CAPTCHAs on the image exactly overlapping the email and password fields (text areas) of the Yahoo iframe below the opaque image. The 'Verify' button on the image of the CAPTCHAs is exactly overlapped with the 'Sign in' button of the Yahoo iframe below.

When an honest user visits `landlordattacker.com` from a vulnerable browser, he solves the two CAPTCHAs on the image to view free content. Since the browser allows

user access to the text area below the image, when the user fills in the CAPTCHA on top, *he actually fills in the username and password of the landlord attacker in the Yahoo iframe below the image.* Once the user clicks the verify button on the image, the 'sign in' button on the Yahoo iframe is clicked instead, thereby logging the user's browser into `www.yahoo.com` as the attacker.

In general, solving a CAPTCHA does not disclose private user information and is perceived as a security feature. Therefore, even a careful user would likely be willing to solve the CAPTCHA. Because the top image is opaque, the user is completely oblivious to the consequences of his seemingly benign action. Once the attacker is logged in from the user's browser, all the potential consequences of login CSRF are possible.

3) User Interaction Interception: This attack is possible due to inconsistent click-event reception. A malicious landlord can launch a user interaction interception attack on his cross-origin tenant by inserting display elements below a cross-origin tenant image. In a webpage containing mutually distrusting principals, each principal's actual content as well as the user interaction with the principal's content are private to that principal (due to the Same Origin Policy). Therefore, the browser must not allow unauthorized observation by a principal on a user's interaction with another tenant.

A malicious landlord attacker can intercept user interaction with an opaque cross-origin image ad with a click event in a browser that gives priority to the user events (such as `onclick`, `onmouseover`) of elements situated below the image. The expected behavior of `onclick` on the image is navigation of user's browser to the advertiser's webpage. A user's interaction with the ad on the malicious landlord's page is private to the advertiser because of the Same Origin Policy. To snoop on the user interaction with the tenant, the landlord fills the entire screen area below the image ad with

buttons that have an `onclick` event defined. If a user visits the landlord's website from a vulnerable browser and clicks on the image ad, the click event of the buttons below the image will be executed. This browser behavior will allow a malicious landlord to monitor user interaction with the honest tenant.

3.3.3 Analysis

Android Mobile, Android tablet on Xoom, Nokia Mini-Map and Opera Mini browsers are susceptible to all the attacks; whereas, none of the desktop browsers are susceptible to any of the attacks. We found discrepancies between browsers made by the same vendors. For instance, while Opera Mini is susceptible to all of the attacks discussed in this section, neither the Opera desktop nor Opera Mobile browsers are vulnerable. However, this behavior does not indicate that Opera Mobile enforces all the same policies implemented in Opera desktop as seen in Section 3.4.

These experiments demonstrate that there are a number of ways in which user actions can be intercepted by hidden and potentially malicious objects when rendered by many popular mobile web browsers. However, as our next set of tests demonstrates, there are more direct ways by which malicious objects can elicit direct user interaction.

3.4 Boundary Control

Many websites contain one or more cross-origin tenants in the form of ads or widgets. Websites (landlord) rely on the browsers to restrict a tenant's dimensions to the display area as defined by the landlord. However, if a browser allows a malicious tenant to control its own dimensions (display ballooning), the tenant can easily expand its own boundaries, completely disregarding the dimensions specified by the cross-origin landlord. *This lack of boundary control allows the tenant to dominate the constrained mobile screen and intercept a user's intended interaction with the landlord.* We discuss details of the discovered vulnerability and then describe potential attacks.

3.4.1 Experimental Evaluation

Mobile and tablet browsers: The Android mobile, iPhone and iPad2 Safari, Opera Mini and Opera Mobile browsers allow an iframe to stretch its own dimensions to fit the content inside the iframe. Even if the landlord specifies the dimensions of the iframe, the cross-origin tenant can change them by putting more content in the iframe. By altering the iframe's dimensions, the tenant's iframe does not alter the layout of the original page; rather all other elements on the screen are adjusted around the new dimensions of the iframe while retaining the original relative layout.

Desktop browsers: We observe that desktop browsers restrict the boundaries of a cross-origin tenant to those defined by the landlord. Instead of expanding, these browsers add scroll bars to the contained iframes, allowing the user to scroll the iframes to access the content not immediately visible due to the boundary restrictions. Therefore, the phishing and password stealing attacks are not possible on desktop browsers.

3.4.2 Attacks

We illustrate two attacks that take advantage of incorrect boundary control.

1) Display Ballooning → Phishing: Display ballooning allows a malicious website principal to push legitimate content far outside of the view of the user (an attack made acute by the general lack of visible scroll bars), thereby causing a client to interact with a seemingly benign but actually dangerous function.

Consider the iGoogle mashup webpage (landlord) containing each widget (tenant) inside an iframe. As shown in Figure 3, an honest user innocently adds a malicious widget (ATTACKER) to his profile. ATTACKER is placed "North" of the honest widget Amazon, which shows online deals and helps the user purchase the items of his choice. The intention of the malicious tenant is to navigate an honest user to a website of the tenant's choice. To launch the attack, the malicious tenant alters his

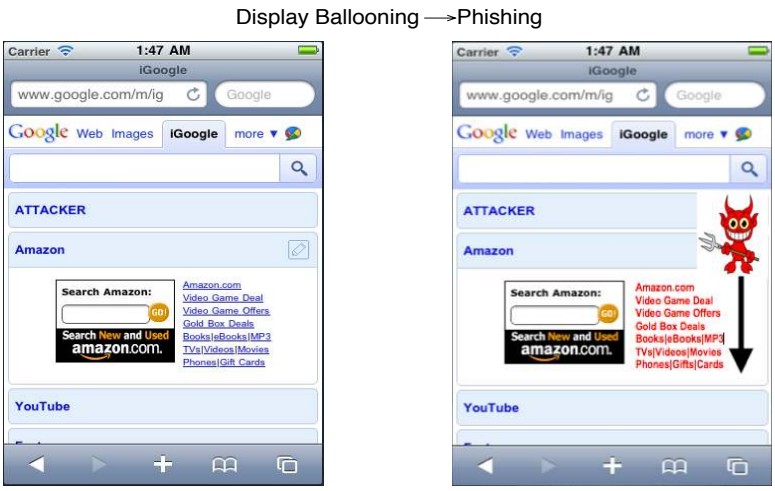

Figure 3: **Left image**: Layout of the malicious and honest widgets on the mashup webpage. 'ATTACKER' is a malicious widget and Amazon and YouTube are honest widgets; **Right image**: The browser allows a cross-origin tenant to write its own dimensions. The malicious widget expands its own dimensions and masquerades as the honest Amazon and YouTube widgets on the browser. It pushes the honest widgets south and launches a phishing attack on the user. This attack works in the iPhone Safari (pictured), Android mobile, iPad2 Safari, Opera Mini and Opera Mobile browsers.

dimensions, expands his own iframe and masquerades as the Amazon and YouTube widgets, while pushing the real Amazon and YouTube widgets "South", far outside of the user's view. Unless the user scrolls down very far, he is unable to notice the attack. The user perceives the masqueraded Amazon as the real widget and clicks on the deals of the attacker's choice.

The tenant attacker does not necessarily need to know the presence and layout of specific widgets on the victim's personal profile. The attacker can masquerade as any of the default widgets generally included on the mashup website. Unless the victim is very familiar with the layout of his profile, he will trust the masqueraded widget. Additionally, if the malicious widget is published on a well known mashup website, a not-so-careful user may be willing to click on links he finds interesting irrespective of the credibility of the widget presenting the links to him. The phishing attack can work on any mash-up website with a similar layout. The Appendix provides code for

a malicious widget on iGoogle.

2) Display ballooning → Password Stealing: Consider a malicious advertisement (tenant attacker) situated to the "North" of the login box of an honest website. The malicious ad can steal a user's credentials by stretching its own dimensions and including a fake login box, which looks exactly the same as the honest website's login box. The real login box would be pushed "South" beyond the bottom of the user's screen. Because the user is not able to see all the content on the screen at the same time, the user will likely enter his credentials in the fake login box.

3.4.3 Analysis

The Android mobile, iPhone and iPad2 Safari, Opera Mini and Opera Mobile browsers are susceptible to phishing and password stealing as a result of display ballooning. The desktop browsers restrict a tenant iframe's dimensions to those specified by the landlord thereby preventing these attacks.

Browsers made by the same vendor deal with boundary control inconsistently. For example, the Opera Mini, Opera Mobile and iPhone Safari browsers exhibit the same vulnerability, whereas their desktop versions do not. Additionally, while the Android tablet browser on Xoom is susceptible to display ballooning similar to its mobile version, the Android tablet browser on Galaxy behaves like desktop browsers, correctly implementing tenant boundary restrictions.

The experiments in Section 3.3 and Section 3.4 demonstrate that none of the desktop browsers are vulnerable to the attacks feasible on mobile browsers. Intuitively, adopting similar policies implemented on desktop browsers will prevent introduction of new vulnerabilities in mobile browsers. However, we show in the next section that reusing desktop browser code without modifications can lead to unexpected vulnerabilities in mobile browsers, due to adjustments made in mobile browser software for improved usability.

3.5 Top Level Frame Navigation

The address bar indicates the URL of the viewed webpage and, in some browsers, the current security status. Because of limited screen real-estate, mobile browsers minimize the address bar once a page is rendered, hiding it from the user. This usability concession in mobile browsers directly conflicts with the 'Top-Level Frame Navigation' display policy [70] implemented throughout desktop browsers. This policy governs a principal's ability to navigate principals of other origins. In particular, this policy allows top-level frames (i.e., the landlord) to be navigated by any of its descendants (i.e., tenants) regardless of their origin. Because users can always see the address bar, it is possible for a user to determine if the current destination represents their intended target or a malicious webpage [70]. Accordingly, all desktop browsers allow a user to *always* view the top-level window's address bar.[5] *We show that since mobile browsers do not make the address bar persistently available to a user, browser policies that assume persistent view of address bar for security can be exploited.* We also discuss the differences in our attack and the already studied attacks [112, 165] that exploit non-persistent address bar in mobile browsers, and argue that our attack is more dangerous and easier to launch.

3.5.1 Attack and Experimental Evaluation

A tenant attacker (descendant) can launch a phishing attack if he can navigate the cross-origin top-level window and the top-level window's address bar is not visible to the user.

Consider a webpage `www.honest.com` consisting of a malicious cross-origin advertisement as shown in Figure 4 (left). The `onload` event of the ad is to navigate the top-level window to `www.attacker.com`, which looks exactly the same as

[5]The Chrome, Firefox and Safari desktop browsers allow users to hide the address bar through options [125].

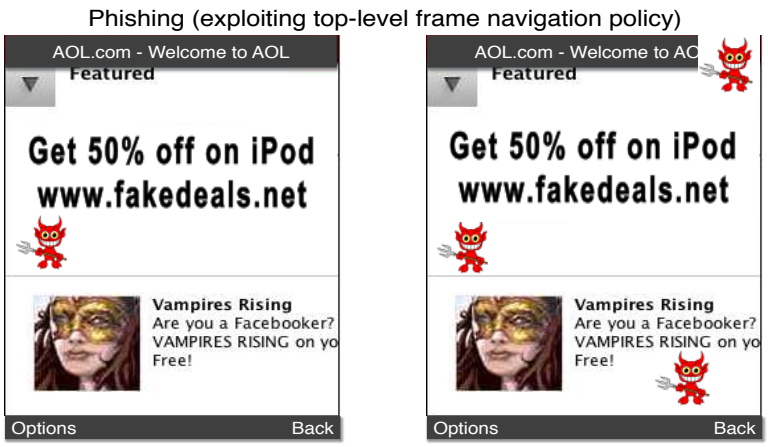

Figure 4: **Left image**: `www.aol.com` webpage containing a cross-origin malicious advertisement. The browser displays only the 'title' of the page and does not display the address bar.; **Right image**: Due to the top-level frame navigation policy, the malicious ad can redirect the top-level window to `www.attacker.com`, which looks exactly the same as AOL's website, thereby launching a phishing attack. The user cannot detect the attack since the address bar containing the URL of the top window is not included in the mobile browser's view due to space constraint. The Nokia Mini-Map and Blackberry Mango browsers are the most susceptible to this attack. However, all other mobile and tablet browsers (except Chrome Beta and iPad2 Safari) are also susceptible to this attack due to address bar not being persistently available while browsing.

`www.honest.com` (Figure 4 (right)) and contains malicious content. When the ad on the honest page is loaded, it navigates the top-level window to the attacker's page. If the user's browser shows the address bar of the top-level window, the user may be able to detect the phishing attack and refrain from interacting with the malicious page. However, if the user's browser does not show the address bar, the user cannot detect the phishing attack. The Appendix provides sample code for this attack.

Experimental Evaluation:

Mobile and tablet browser results: All ten mobile and three tablet browsers allow a tenant principal of any origin to navigate the top-level window to any source.

The iPhone Safari browser minimizes the top-level address bar for better usability once a page is rendered. Moreover, the address bar disappears from view once a user

starts interacting with the content on the page. This behavior is seen in all mobile browsers except Blackberry Mango, Chrome Beta, IE Mobile 8 and Nokia Mini-Map. IE Mobile browser persistently displays the address bar only in the portrait mode and never in the landscape mode. The Chrome Beta is the only mobile browser allowing persistent view of the address bar. In the Blackberry Mango and Nokia Mini-Map browsers, the address bar of the top-level window is *never* accessible to the user on the screen while browsing. The web address of the top-level window can be viewed from Options → Advanced → Page Info in the Nokia Mini-Map browser. In the Blackberry Mango browser, a user is required to click on the lock icon in the top right corner of the screen to access the address of the webpage. It is difficult for a user to browse to this page info every time he wants to access the top level URL. This makes the Blackberry Mango and Nokia Mini-Map browsers the most susceptible to phishing attacks by navigation of top-level window to malicious pages, since the user can never detect the attack unless he intentionally checks the page information and views the webpage's address.

Interestingly, Safari on the iPad2 differs slightly from its iPhone version in that the address bar is present at all times, enabling users to protect themselves from the phishing attack. However, the Android tablet browsers (both Xoom and Galaxy) exhibit similar behavior as their mobile version and hide the address bar when a user starts interacting with the webpage. Therefore, the Android tablet browsers are susceptible to the phishing attack. We also note that due to the smaller screen size of mobile browsers, the complete URL of a webpage is not necessarily displayed to a user. This makes it even more difficult for a user to make a decision of the credibility of a website at the time of page load, when the address bar temporarily flickers at the top of the browser.

Desktop browser results: All five desktop browsers allow a tenant principal of

any origin to navigate the top-level window to any source. However, the desktop browsers *always* display the address bar in the window. We note that if Chrome's option to hide the address bar becomes the widespread default, the 'Top-Level Frame Navigation' policy should be reconsidered for all browsers.

3.5.2 Analysis

When a user interacts with a webpage on a desktop browser by scrolling or zooming, the top-level address bar is always available to the user. However, because of the drastically reduced screen size of mobile devices, removing the address bar from view makes sense in mobile browsers. Because this necessarily pushes the address bar out of the user's sight for most of the time while browsing, the current policy for top-level frame navigation is not appropriate for mobile browsers. We discuss potential solutions to this problem in Section 6.4.3.

We note that our phishing attack is significantly different than the existing phishing attacks [112, 165, 183] exploiting address bar hiding in mobile browsers. The existing attacks [112, 165] assume that the user is already on a phishing website, spoof the address bar and then preclude the user from viewing the 'real' address bar using Javascript. Therefore, a successful attack requires an attacker to trick a user into browsing to the phishing website. Our attack does not assume that a user is already on a phishing website. Instead, an attacker can post an advertisement on *any* legitimate website and then redirect the user to a phishing website without requiring any explicit user interaction. This makes our attack more dangerous and feasible as compared to the attacks that require user interaction to launch a phishing website. Any legitimate website hosting cross-origin content becomes vulnerable to our attack. We note that once an attacker redirects a user to a phishing website by exploiting the

Table 3: Summary of observed display-related vulnerabilities in candidate browsers and respective attacks possible (A ✓depicts that attack is possible). **1**) Equivalent vulnerabilities exist in mobile and tablet browsers with different rendering engines. **2**) Mobile, tablet and desktop browsers from the same vendor do not necessarily implement the same code to handle display elements in different settings. **3**) Desktop browsers are more compliant with security policies for display.

Type	**Rendering Engine**	**Browser Name**	**Attacks**		
			Vulnerability - Incorrect handling of user access to overlapping elements	*Vulnerability* - Cross-origin tenant modifying self dimensions	*Vulnerability* - Inconsistent view of address bar
			Click fraud, Login CSRF, User Interaction Interception	Display Ballooning: Password Stealing, Phishing	Phishing
Mobile	Webkit	Android	✓	✓	✓
		Blackberry Webkit			✓
		Chrome Beta			
		iPhone Safari		✓	✓
		Nokia Mini-Map	✓		✓
	Presto	Opera Mini	✓	✓	✓
		Opera Mobile		✓	✓
	Gecko	Firefox Mobile			✓
	Mango	Blackberry Mango			✓
	Trident	Internet Explorer			✓
Tablet	Webkit	Android on Xoom	✓		✓
		Android on Galaxy			✓
		Safari on iPad		✓	
Desktop	Presto, Gecko, Webkit Trident	Opera, Firefox, Safari, Chrome, Internet Explorer			

top-level frame navigation policy, existing address bar spoofing techniques [112, 165] can be used to increase the success rate of the attack.

3.6 Discussion and Potential Solutions

Mobile browsers necessarily make considerations for the constrained platform on which they run. Unfortunately, in the process of porting their software to these devices, vendors have introduced a number of new classes of vulnerabilities. While seemingly unrelated, Table 3 shows that these issues are repeated across many mobile browser vendors. The vulnerabilities presented in this work are made even more dangerous by the constrained nature of the mobile screen as shown in Section 3.4.2.

A subset of vendors of the evaluated browsers have confirmed the presence of the vulnerabilities [2–4]. We note that unavailability of a standard for user event

routing and boundary control may be a cause of these vulnerabilities. Identical vulnerabilities were observed in browsers irrespective of the rendering engine used or the manufacturer. For example, the Android Mobile (Webkit) and Opera Mini (Presto) browsers exhibit the same issues; whereas, the five Webkit-based mobile browsers do not demonstrate all of the same vulnerabilities. Intuitively, assuming that browsers built by the same company have some overlap in the development teams suggests that browser components may be reused across platforms. However, the differences in the presence of vulnerabilities in the mobile, tablet and desktop browsers built by the same vendor (e.g., Opera) suggests that new vulnerabilities have been introduced while porting components from existing browser software to a new platform. Whether the discovered vulnerabilities are implementation or design errors in individual browsers is hard to state with certainty. The pervasive nature of the vulnerabilities hints at a more concerning trend.

We propose solutions for the vulnerabilities discussed in this chapter. Browsers should always route the click, hover and write user events exclusively to the top element in a stack of overlapped elements. This will provide consistency in handling user event routing and also prevent the attacks discussed in Section 3.3.2. Secondly, The attacks possible due to erroneous boundary control can be prohibited by restricting dimensions of tenant iframes to those specified by the landlord irrespective of the origins of the tenant and landlord. We note that the evaluated desktop browsers have implemented preventive measures against the attacks discussed herein. We recognize that if desktop browsers implemented the exact same erroneous policies discussed in this work, they might be susceptible to the attacks described herein. However, it is important to note that in addition to desktop browsers implementing the respective policies correctly, the constrained screen size of mobile devices makes the attacks discussed in this work more plausible and dangerous on mobile browsers. For example, the non-persistent nature of address bar and small screen size makes the phishing

attacks discussed in Sections 3.4 and 3.5 more difficult to detect and easier to launch.

Although borrowing desktop browser policies addresses the vulnerabilities in user event routing and boundary control, the small screen size of mobile devices demands more restrictive policies than those implemented in desktop browsers to prevent the phishing attack discussed in Section 3.5. We propose using Gazelle's top-level frame navigation policy [209] allowing only tenants with the same origin and the user to navigate the top-level window. This approach would better balance issues of usability, specifically screen real-estate, and security. A more extreme solution would be removing support for the top-level frame navigation policy from mobile browsers; however, legitimate webpages relying on this mechanism for functionality may break. Offloading security decisions to the cloud [62] would be another alternative solution to the generic problem of tension between security and usability on small mobile screens. Most critically, borrowing the top-level frame navigation policy to the mobile environment is evidence that security and usability teams are not interacting closely enough with each other. Any solutions should be applied with input from both groups.

We evaluated the impact of the discovered vulnerabilities on formally defined display security policies in earlier works. One of the recent works on display security in desktop browsers that solved some well-known issues is the Gazelle browser work by Wang et al. [209]. Gazelle is a secure web browser [209] that defines more formal access control rules for position, dimension, content and content source (location) control between cross-origin principals. According to Gazelle's display access control matrix (refer to the Ideal column in Table 4 and Table 5), a landlord should be able to read and write the position and dimensions of a tenant. The landlord can write over the entire tenant screen area with new content (change location) without modifying the current tenant's private content. However, he should not be able to read or write a cross-origin tenant's private content (pixels). The tenant should be able to read and write its own pixels and read its own dimensions.

We tested the candidate mobile and desktop browsers against these proposed policies. Our experiments showed that the desktop browsers conform to all of the suggested policies. However, the mobile and tablet browsers violated recommended display security policies due to the new found vulnerabilities in user event routing and boundary control. The mobile browsers susceptible to the 'user interaction interception' attack allow a landlord to read the cross-origin tenant's private information (user interaction) as shown in Table 4. Moreover, the mobile browsers allowing 'display ballooning' allow a tenant to write its own dimensions (Table 5). This shows that the new found vulnerabilities violate formally specified display security policies.

Table 4: Gazelle's (the ideal column) policies for a landlord to access a cross-origin tenant's position, dimensions, pixels and URL location. R: Read access. W: Write access. **R***: Android mobile, Android tablet on Xoom, Nokia Mini-Map and Opera Mini browsers allow a landlord to read user interaction with its cross-origin tenant. This vulnerability breaches the access control policy for the tenant's 'pixels' allowing the landlord to launch the user interaction interception attack.

	Landlord			
	Ideal [209]	**Fail**		**Pass**
		Android mobile, Android tablet on Xoom, Opera Mini	Nokia Mini Map	Android tablet on Galaxy, Chrome Beta, iPhone and iPad2 Safari, IE Mobile, Opera Mini and Mobile, Blackberry Mango and Webkit
position (x,y,z)	*RW*	RW	RW	RW
dimensions (height, width)	*RW*	RW	RW	RW
pixels	–	**R***	**R***	–
URL location	*W*	W	–	W

The relevance of our observations goes well beyond web browsing. A significant amount of research effort has recently focused on the security of mobile applications [102, 105, 106]. These studies have generally centered around applications built for specific platforms. However, an increasing number of applications are becoming highly dependent on the browser. In particular, applications by a number of popular companies (e.g., BBC) are actually wrappers around the browser and point their users to specific webpages within a target domain. The advantage to this approach

Table 5: Gazelle's (the ideal column) policies for a tenant of a cross-origin landlord to access its own position and dimensions on the landlord's page, its URL location and its pixel content. R: Read access. W: Write access. **RW***: Android mobile, iPhone and iPad2 Safari, Opera Mini and Opera Mobile allow a cross-origin tenant to write self dimensions and are thus susceptible to the phishing and password stealing attacks. IE Mobile does not allow a tenant to read self dimensions. This may allow a malicious landlord to shrink the tenant's dimensions to 'zero' without any notice to the tenant.

	Tenant			
		Fail	**Pass**	
	Ideal [209]	Android mobile, iPhone and iPad2 Safari, Opera Mini and Mobile	Android tablet on Galaxy and Xoom, Chrome Beta, Firefox Mobile, Nokia Mini-Map, Blackberry Mango and Webkit	IE Mobile
position (x,y,z)	–	–	–	–
dimensions (height, width)	R	**RW ***	R	–
pixels	RW	RW	RW	RW
URL location	RW	RW	RW	RW

is that it allows companies to ensure a relatively consistent user experience across all platforms with minimal development effort. As a consequence, however, such "applications" now also potentially become vulnerable to the kinds of attacks discussed in this chapter.

3.7 Conclusion

Constrained screen size fundamentally changes the browsing experience on mobile phones. Crowded layout, the inability to consume large amounts of content concurrently and the difficulty in discerning boundaries between different objects on a webpage make it hard for users to browse the web in the manner to which they are accustomed. In response to these problems and to alleviate these difficulties, mobile browsers have been changed significantly from their desktop counterparts. However, the impact of these changes on security has not been studied. In this work, we perform the first large-scale comparison of display security between the most popular mobile, tablet and desktop browsers and demonstrate that the differences are far from simply

cosmetic. We identify and implement a number of attacks based on two new classes of vulnerabilities found only on mobile and tablet browsers, and then present solutions to address the vulnerabilities. We then identify a third class of vulnerability that exploits the small screen size of mobile devices and a universally implemented policy in all browsers. Our results and feedback from browser vendors exemplify that new vulnerabilities have been introduced while porting browser software to mobile platforms and that usability should be considered while designing solutions instead of blindly porting desktop browser code to the mobile environment.

3.8 Appendix

A] Code Samples:

1) Login CSRF attack (Section 3.3.2): landlordattacker.com

```
<html>
<body>
  <IFRAME NAME="csrf" SRC="http://mlogin.yahoo.com/w/login/user?ssl=false"
    style="position:absolute; top:20; left:10;" WIDTH="250" HEIGHT="100"></IFRAME>
  <img src="image.png" WIDTH="400" HEIGHT="400"
    style="position:absolute; top:20; left:3;">
</body>
</html>
```

Image.png is the image shown in Figure 2 (left). The image totally overlaps the iframe below it.

2) Display ballooning attack (Section 3.4.2): Widget code

```
<?xml version="1.0" encoding="UTF-8" ?>
<Module>
  <ModulePrefs title="ATTACK" height="25" width="10" scaling="true" />
  <Content type="html">
```

```
  <![CDATA[
  <div style="text-align:center"> <a>
    <IFRAME src="http://attacker.html" width="100\%" height="110" frameborder="0"
      style="position:absolute; top:0px; left:0px"></IFRAME>
  </a> </div>
  ]]>
  </Content>
</Module>
```

The attacker.html iframe contains the malicious imitated widgets for Amazon and YouTube.

3) Phishing attack (Section 3.5.1)

1) *Parent code:*

```
<html>
  <head> Legitimate parent </head>
  <body>
    <IFRAME src="http://www.evil-advertisement.html" WIDTH=300 HEIGHT=300></IFRAME>
  </body>
</html>
```

2) *Cross-origin child (e.g., http://www.evil-advertisement.html) code:*

```
<html>
  <head> Cross-origin advertisement
    <script>
    function navigate_parent() {
      window.top.location="http://www.phishing-attack.com";
    }
    </script>
  </head>
```

```
  <body onload="javascript:navigate_parent()"> </body>
</html>
```

The parent includes a malicious advertisement from `http://www.evil-advertisement.html`, which in turn navigates the top-level window of the browser to `http://www.phishing-attack.com` that looks exactly like the original parent webpage.

CHAPTER IV

AN EMPIRICAL EVALUATION OF SSL INDICATORS IN MOBILE BROWSERS

4.1 Introduction

Mobile browsers provide a rich set of features that often rival their desktop counterparts. From support for Javascript and access to location information to the ability for third-party applications to render content through WebViews, browsers are beginning to serve as one of the critical enablers of modern mobile computing. Such functionality, in combination with the near universal implementation of strong cryptographic tools including SSL/TLS, allows users to become increasingly reliant upon mobile devices to enable sensitive personal, social and financial exchanges.

In spite of the availability of SSL/TLS, mobile users are regularly becoming the target of malicious behavior. A 2011 report indicates that mobile users are three times more likely to access phishing websites than desktop users [80]. Security indicators (i.e., certificate information, lock icons, cipher selection, etc.) in web browsers offer one of the few defenses against such attacks. A user can view different security indicators and related certificate information presented by the browser to offer signals or clues about the credibility of a website. Although mobile and tablet browsers appear to support similar security indicators when compared to desktop browsers, the reasons behind the increasing number of attacks on mobile browsers [24, 31] are not immediately clear.

In this chapter, we perform the first comprehensive empirical evaluation of security indicators in mobile web browsers. The goal of this work is *not* to determine if average users take advantage of such cues, but instead whether security indicators are applied

in a manner that allows expert users to accurately determine the identity of a website or verify the use of strong cryptographic primitives for communications. We believe that this distinction is critical because it highlights areas where not even the best trained users will be able to differentiate between malicious and benign behavior. Rather than an ad hoc analysis, we base our study on the recommendations set forward by the W3C for user interface security [35] as a proxy for best practices. In particular, we measure which browsers strictly conform to the absolute requirements ("MUST" clauses) and prohibitions ("MUST NOT" clauses). We perform our analysis across ten mobile and two tablet browsers, representing greater than 90% of the mobile market share [39], and then compare our results against the five most popular desktop browsers. Our experiments demonstrate that while the majority of desktop browsers largely meet the W3C recommendations, all mobile browsers fail to meet many of the guidelines. Additionally, we observe that mobile browsers exhibit tremendous inconsistency in the presentation and availability of such indicators in contrast to traditional desktop browsers.

Our main contribution is a comprehensive and systematic evaluation and comparison of security indicators and security information for mobile and tablet browsers, to our knowledge the first such analysis undertaken. The main findings of our experiments are that all popular mobile and tablet browsers fail to meet, in numerous instances, the recommendations in the W3C guidelines for user interface of security information, whereas in comparison desktop browsers largely follow the guidelines. Our findings of tremendous inconsistency of user interfaces across mobile browsers, and between mobile and desktop browsers, are also expected to be of considerable interest. Among other contributions, we outline attacks on mobile browsers, such as phishing and undetectable man-in-the-middle, enabled by failure to properly follow these guidelines; and we highlight missing security indicators, e.g., extended validation (EV) SSL indicators [34, 136, 192]. These are intended to convey an augmented

assurance process, however we find most mobile browsers fail to implement EV-SSL indicators visible to users, and their absence along with that of any distinguishing browser behavior, precludes EV-SSL certificates from providing relying parties any benefits beyond non-EV SSL certificates.

The remainder of our chapter is organized as follows: Section 4.2 provides definitions and explains the mandatory elements of the W3C guidelines; Section 4.3 provides the primary results of our evaluation; Section 4.5 discusses secondary observations; Section 4.4 presents ways in that a user can be mislead about the identity of a website or the use of encryption and attacks that are enabled by this confusion; Section **??** presents an overview of related research; and Section 5.7 offers a discussion of our findings and concluding remarks.

4.2 Background on the W3C Recommendations

The World Wide Web Consortium (W3C) has defined user interface guidelines [35] for the presentation and communication of web security context information to end-users of both desktop and mobile browsers. For context in later sections, we first define the terminology and then provide a brief explanation of the W3C guidelines referenced within this chapter.

4.2.1 Definitions

User interface elements: User interface elements in browsers are divided in two categories [35]:

- *Primary User Interface:* the portions of a user interface that are available to users without being solicited by a user interaction. The primary user interface elements related to security traditionally include the padlock icon, the address bar, the `https` URL prefix, the favicon, and the site-identity button or URL coloring to signify the presence of EV-SSL and SSL certificates [34].

- *Secondary User Interface:* the portions of a user interface that are available to the user after they are solicited by a specific user interaction. The secondary user interface elements related to security include the security properties dialog, domain name, owner information, verifier information, information on why a certificate is trusted, validity period of manually accepted certificates (self-signed) and cipher details of an SSL connection.

Trust anchor: A trust anchor represents an authoritative entity represented by a public key and associated data. The public key is used to verify digital signatures and the associated data is used to constrain the types of information for which the trust anchor is authoritative. Relying parties (web browsers) use trust anchors to determine if digitally signed information objects are valid by verifying digital signatures using the trust anchor's public key and by enforcing the constraints expressed in the associated certificate data. Our interpretation is that a trust anchor refers to a certificate authority (CA).

Root: A root is a trust anchor that is any certificate authority (CA).

Trusted root: A trusted root is a CA whose public key is a priori trusted by the browser and may certify other keys.

Certificates: Public key certificates are widely used to provide keying material and convey a website's identity information to the user. The W3C defines four types of certificates. We define two additional certificate types that are not covered in the W3C document. We provide our interpretation for the definitions of certificate types in the W3C document where they are ambiguous. For additional information regarding the commercial practice of issuing and managing SSL certificates, please

refer to the requirements defined by the CA/Browser forum [36].

- *Validated certificate:* This is a public key certificate that has been verified by chaining up to a trusted root. Our interpretation is that a standard SSL certificate signed by a CA trusted by a browser refers to a validated certificate.

- *Augmented assurance certificate:* The certificate chain for such a certificate MUST be validated up to a trusted root that is recognized as augmented assurance qualified by the user agent (user's browser). We interpret an EV-SSL certificate as an augmented assurance certificate that is validated by the browser.

- *Self-signed certificate and untrusted root certificate:* A self-signed certificate is a certificate that is signed by its own creator and is not a priori trusted by a browser. Our interpretation of an untrusted root certificate is that it refers to a certificate holding the public key of a CA, that is signed by a CA not a priori trusted by the user's browser.

- *Interactively accepted trust anchors or certificates:* This refers to either a CA or a website's public key that is accepted by a user and thereby used as a trust anchor by the browser. Whether the trust anchor is accepted just for the present transaction or for the present and the future transactions depends on the options presented to the user by the browser and then the option chosen by the user.

- *Unverifiable certificates:* An unverifiable certificate refers to a certificate that is neither included in a user's browser, nor can be verified by the browser through a trust chain.

- *Untrusted site certificates:* These certificates are signed by a trust anchor that is not a root and the trust anchor is not included in the user's browser.

When a browser receives a website certificate, the public key therein (and the certificate) is untrusted unless either the certificate was previously interactively accepted (for future sessions), or trust can be derived in it transitively, through a trust chain starting from a trust anchor (i.e., a CA key already trusted by the browser).

Pinning: Pinning associates one or more certificates with a specific website. The certificate provided by the website can either be self-signed or one issued by an untrusted root. Once a user interactively accepts such a certificate for the first time, the browser pins the certificate to the website. After pinning, the browser warns users only when the same website presents a different certificate. No warning messages are shown by the browser if a site shows a certificate consistent with previously pinned certificates for that site.

Identity Signal: An identity signal on a TLS-secured webpage includes information about the owner of the webpage and the certificate issuer's organization. A webpage's certificate provides its owner information and the issuer's (e.g., Certificate Authority) organization.

De-referencing a URI: The act of retrieving a representation of an information resource identified by a URI is known as dereferencing that URI [27]. The act of creating a representation is simply a transformation of information into an appropriate form consumable by a user. For example, when a user accesses a bank account statement, the binary data is retrieved from a database on the server and presented to the user after conversion into a readable text format.

TLS/SSL: The Transport Layer Security (TLS) protocol allows client/server applications to communicate across a network in a way designed to prevent eavesdropping and tampering. The predecessor protocol of TLS is called Secure Socket Layer (SSL).

Strong TLS: An `http` transaction is strongly TLS-protected if it is TLS-protected, an `https` URL was used, strong TLS algorithms were negotiated for both confidentiality and integrity protection, and at least one of the following conditions is true: the server used a validated certificate that matches the dereferenced URI; the server used a self-signed certificate that was pinned to the destination; the server used a certificate chain leading to an untrusted root certificate that was pinned to the destination.

A strong TLS algorithm implies that no version of the TLS protocol that suffers known security flaws has been negotiated. Therefore, versions of SSL prior to SSLv3 MUST NOT be considered strong. Additionally, a strong TLS algorithm must also select a cipher suite for which key and algorithm strengths correspond to industry practice. More information on strong and weak TLS algorithms can be found in the W3C document [35] and RFC 4346 [26].

Weak TLS: An `http` transaction is weakly TLS-protected if it is TLS-protected, but strong TLS protection could not be achieved for one of the following reasons: TLS handshake used an anonymous key exchange algorithm, such as DH_anon; the cryptographic algorithms negotiated are not considered strong, such as TLS_KRB5_EXPORT_WITH_DES_CBC_40_SHA; certificates were used that are neither validated certificates nor self-signed certificates pinned to the destination.

Error messages: The W3C document defines common error interaction requirements and practices to signal two classes of errors ordered by increasing severity: warning/caution messages and danger messages.

Warning/caution messages are intended for situations when the system has reason to believe that the user may be at risk based on the current security context information, however a determination cannot positively be made. Danger Messages are intended for situations when there is a positively identified danger to the user (i.e., not merely a risk).

Subject organization logotype: This is a logotype representing the organization identified in the subject name in the certificate.

4.2.2 W3C Guidelines

We chose a subset of the absolute requirements (MUST) and prohibitions (MUST NOT) specified in the W3C user interface guidelines.[1] We omitted the guidelines represented by clauses including the MAY, MAY NOT, SHOULD and SHOULD NOT keywords as they represent the optional guidelines [25]. We classify the W3C guidelines into *five* categories: identity signal, certificates, TLS indicators, robustness and error messages.

1) Identity signal: availability:

The security indicators showing identity of a website MUST be available to the user either through the primary or the secondary interface at all times.

2) Certificates: required content:

In addition to the identity signal, the web browsers MUST make the following security context information available through information sources (certificates): the webpage's domain name and the reason why the displayed information is trusted (or not).

3) TLS indicators:

[1]The guidelines deemed to be the most critical and definitively testable were selected based on the authors' experience and knowledge of the area of SSL indicators.

Table 6: Details of the browsers used for experimental evaluation. (*: The version numbers of these browsers were not apparent. We have used the default browsers shipped with the referenced version of the OS.)

Category	Browser Name	Version	Rendering Engine	Operating System	Device
Mobile	Android	2.3.3	Webkit	Android 2.3.3	Nexus One
	Blackberry	5.0.0	Mango	Blackberry OS 5.0.0.732	Bold 9650
	Blackberry	6.0.0	Webkit	Blackberry OS 6	Torch 9800
	Chrome Beta	0.16.4130.199	Webkit	Android 4.0.3	Nexus S
	Firefox Mobile	4 Beta 3	Gecko	Android 2.3.3	Nexus One
	Internet Explorer Mobile	*	Trident	Windows Phone 7.0.7004.0 OS	LG-C900
	Nokia Browser	7.4.2.6	Webkit	Symbian Belle	Nokia 701
	Opera Mini	6.0.24556	Presto	Android 2.3.3	Nexus One
		5.0.019802	Presto	iOS 4.1 (8B117)	iPhone
	Opera Mobile	11.00	Presto	Android 2.3.3	Nexus One
	Safari	*	Webkit	iOS 4.1 (8B117)	iPhone
Tablet	Android	*	Webkit	Android 3.1	Samsung Galaxy
	Safari	*	Webkit	iOS 4.3.5 (8L1)	iPad 2
Desktop	Chrome	15.0.874.106	Webkit	OS X 10.6.8	–
	Firefox	7.0.1	Gecko	OS X 10.6.8	–
	Internet Explorer	8.0.7600.16385	Trident	Windows 7	–
	Opera	11.52	Presto	OS X 10.6.8	–
	Safari	5.1.1	Webkit	OS X 10.6.8	–

a) Significance of presence: Any UI indicator (such as the padlock) MUST NOT signal the presence of a certificate unless all parts of the webpage are loaded from servers presenting at least validated certificates over strongly TLS-protected interactions.

b) Content and Indicator Proximity: Content MUST NOT be displayed in a manner that confuses hosted content and browser chrome indicators, by allowing that content to mimic chrome indicators in a position close to them.

c) Availability: The TLS indicators MUST be available to the user through the primary or the secondary interface at all times.

4) Robustness: visibility of indicators:

Web content MUST NOT obscure the security user interface.

5) Error messages:

a) Interruption: Both warning/caution and danger messages MUST interrupt the user's current task, such that the user has to acknowledge the message.

b) Proceeding options: Warning/caution messages MUST provide the user with distinct options for how to proceed (i.e., these messages MUST NOT lead to a situation in which the only option presented to the user is to dismiss the warning and continue).

c) Inhibit interaction: The interactions for danger messages MUST be presented in a way that makes it impossible for the user to go to or interact with the destination website that caused the danger situation to occur, without first explicitly interacting with the danger message.

4.3 Empirical Observations

We evaluate ten mobile and two tablet browsers against the W3C recommended practices for security indicators. The details of the browsers are provided in Table 6. For each of the guidelines described in Section 4.2.2, we create and run a set of experiments to verify compliance on all the candidate browsers and record our observations. All the experiments were performed on web browsers on real mobile phones, and are recreated in the respective emulators to generate many of the figures throughout the chapter. The browser versions used in our evaluation are approximately the latest as of February 12^{th}, 2012. Table 7 through Table 12 provide the synopsis of the results of our experiments.

4.3.1 Identity Signal: Availability

An identity signal contains information about the owner of a website and the corresponding certificate issuer. Before issuing a certificate, the certificate provider requests the contact email address for the website from a public domain name registrar,

Table 7: Results of experiments on candidate mobile browsers to test compliance with the first two W3C guidelines given in Section 4.2.2. Each guideline column consists of sub-columns stating the experiments performed on the browsers. An × implies that the browser does not comply with the respective W3C guideline. A · implies that the browser complies with the respective W3C guideline.

Mobile and Tablet Browsers (See Table 6 for versions)	**1) Identity signal: availability**		**2) Certificates: required content**	
	Owner information available?	Certificate issuer's info available?	Domain name available?	Information on why certtificate trusted available?
Android	•	•	•	×
Blackberry Mango	•	•	•	•
Blackberry Webkit	•	•	•	•
Chrome Beta	•	•	•	×
Firefox Mobile	•	•	•	×
iPhone Safari	×	×	×	×
Nokia Browser	•	•	•	×
Opera Mini	×	×	×	×
Opera Mobile	×	×	×	×
Windows IE Mobile	×	×	×	×
Safari on iPad 2	×	×	×	×
Android on Galaxy	•	•	•	×

Table 8: Results of experiments on traditional web browsers to test compliance with the same guidelines as given in Table 7, with × and · symbols also implying the same. Note that all the desktop browsers are compliant.

Desktop Browsers (See Table 6 for versions)	**1) Identity signal: availability**		**2) Certificates: required content**	
	Owner information available?	Certificate issuer's information available?	Domain name available?	Information on why cert trusted available?
Chrome	•	•	•	•
Firefox	•	•	•	•
IE	•	•	•	•
Opera	•	•	•	•
Safari	•	•	•	•

and checks that published address against the email address supplied in the certificate request. Therefore, the owner of a website is someone in contact with the person who registered the domain name. Popular browsers represent the owner information of a website using different terminology including owner, subject, holder and organization.

We visited a public webpage presenting a trusted root certificate from all the candidate browsers. We then evaluated the browsers for the presence of identity signal, either on the primary or the secondary interface.

Figure 5: Identity information displayed by Firefox Mobile.

Observations: The IE Mobile, iPhone and iPad Safari, and Opera Mini and Mobile browsers do not provide a user interface to view certificates. Accordingly, the identity signal information is not available for a user of these browsers and thus none of these five browsers comply with the W3C guideline for availability of identity signal. We note that when a website presents a certificate that is from a CA not from a trusted root, all the browsers provide an interface to view the certificate via an error message. The Android mobile and tablet, Blackberry Mango and Webkit, Chrome Beta and Nokia browsers always allow a user to view certificates (both trusted and untrusted) and therefore comply with this guideline. A user is required to click the lock icon to view certificate information on the Chrome Beta and Blackberry Mango browsers. However, the browsers do not provide any visual indication to the user about this process of accessing the certificate information. Browsers supporting a UI for viewing certificate information provide a clear indication in the "options" in the browser menu. Although the Firefox Mobile browser does not support a certificate UI, it displays the identity information of a website when the site identity button is clicked, as shown in Figure 5. All desktop browsers comply with this guideline. Tables 7 and 8 provide the summary of our results.

4.3.2 Certificates: Required Content

In addition to the identity signal content, a certificate from a website must provide the same website's domain name and the reason why the displayed information is trusted (or not). Trust reasons include whether or not a certificate was accepted

interactively, whether a self-signed certificate was used, whether the self-signed certificate was pinned to the site that the user interacts with, and whether trust relevant settings of the user agent were otherwise overridden through user action. We believe that information such as "certificate is implicitly trusted" and "the certificate chain is trusted/valid" also conveys the reason behind a browser trusting or not trusting a particular website.

We analyzed the candidate browsers for the presence of the required certificate content by visiting a website that uses strongly TLS-protected connection with its clients.

Observations: The IE Mobile, iPhone and iPad Safari, and Opera Mini and Mobile browsers do not provide a user interface to view certificates from trusted CAs. Therefore, these browsers fail to meet the W3C guideline. Additionally, even though the remaining mobile and tablet browsers provide a user interface to view certificate information, they do not provide an explanation on why a particular certificate is trusted. Only the Blackberry Mango and Webkit browsers comply with the guideline by making all the required parts of a certificate available. When a website presents a certificate from a trusted CA, the Blackberry Mango and Webkit browsers show the reason "certificate is implicitly trusted". Therefore, all but two mobile and tablet browsers fail to meet this W3C guideline. All desktop browsers follow this guideline correctly. Tables 7 and 8 provide the summary of our results.

4.3.3 TLS Indicators

TLS indicators include the `https` prefix, the padlock icon, information about the ciphers used in the connection and url coloring (or site identity button) to depict the

Table 9: Results of experiments on candidate mobile browsers to test compliance with the W3C guidelines 3a and 3b given in Section 4.2.2. The symbol notation is as defined in Table 7. 's': Implies that the `https` URL prefix is present on the 's'econdary interface.

Mobile and Tablet Browsers (See Table 6 for versions)	**TLS indicators**		
	3a) significance of presence		**3b) position**
	Mixed content: no lock shown?	Mixed content: no `https` shown?	Favicon not next to lock icon?
Android	Open lock with a question mark	×	×
Blackberry Mango	×	×	•
Blackberry Webkit	×	×	•
Chrome Beta	Closed lock with a cross on top	https striked through	•
Firefox Mobile	No security indicators shown	×	•
iPhone Safari	•	×	•
Nokia Browser	•	×	•
Opera Mini	•	×	•
Opera Mobile	•	×	•
Windows IE Mobile	×	×	•
Safari on iPad 2	•	×	•
Android on Galaxy	Open lock with a question mark	×	•

Table 10: Results of experiments on traditional web browsers to test compliance with the same guidelines as Table 9. The symbol notation is as defined in Table 7. 'p': Implies that the `https` URL prefix is present on the 'p'rimary interface.

Desktop Browsers (See Table 6 for versions)	**TLS indicators**		
	3a) significance of presence		**3b) position**
	Mixed content: no lock shown?	Mixed content: no `https` shown?	Favicon not next to lock icon?
Chrome	Lock with a yellow triangle	×	•
Firefox	•	×	•
IE	•	×	•
Opera	•	×	•
Safari	•	×	•

difference between EV-SSL and SSL certified webpages.

a) Significance of presence: If a web browser displays a TLS indicator for the presence of a certificate for a webpage consisting of content obtained over both `http`

Figure 6: Blackberry Mango browser rendering a mixed content webpage. Note that the webpage contains a Google map obtained over an http connection. Although the webpage holds mixed content, the browser displays the padlock icon as well as the https URL prefix indicators. This behavior fails to meet with guideline 3a.

and `https` connections (mixed content), this guideline is not followed.

We created a simple webpage that uses a strong TLS connection to retrieve the top level resource and embedded a map obtained from a third-party over an unsecured `http` connection. We analyzed the browsers while rendering the this page for two basic TLS security indicators: the `https` URL prefix and the padlock icon. If a browser shows any of these two indicators on a mixed content webpage, it does not follow the W3C guideline. We also observed whether a browser shows a warning to the user suggesting the presence of mixed content on the webpage.

Observations: The Blackberry Mango, Blackberry Webkit and IE Mobile browsers display a lock icon on a webpage holding mixed content, thus failing to meet the W3C guideline. Figure 6 shows a screen shot of the Blackberry Mango browser when a mixed content webpage is rendered. The Blackberry Webkit and IE Mobile browsers display a mixed-content warning and, if the user proceeds to the webpage, a lock icon is displayed. The Android browsers on the mobile and tablet devices present an open lock with a question mark inside the lock. The Chrome Beta browser displays a closed lock with a cross on top and a striked through https URL prefix for a mixed content webpage. This behavior of Android and Chrome is inconsistent with the other browsers. Therefore, it is necessary for the users of these browsers to understand the meaning of the new symbols in order to interpret its reference to mixed content on a webpage.

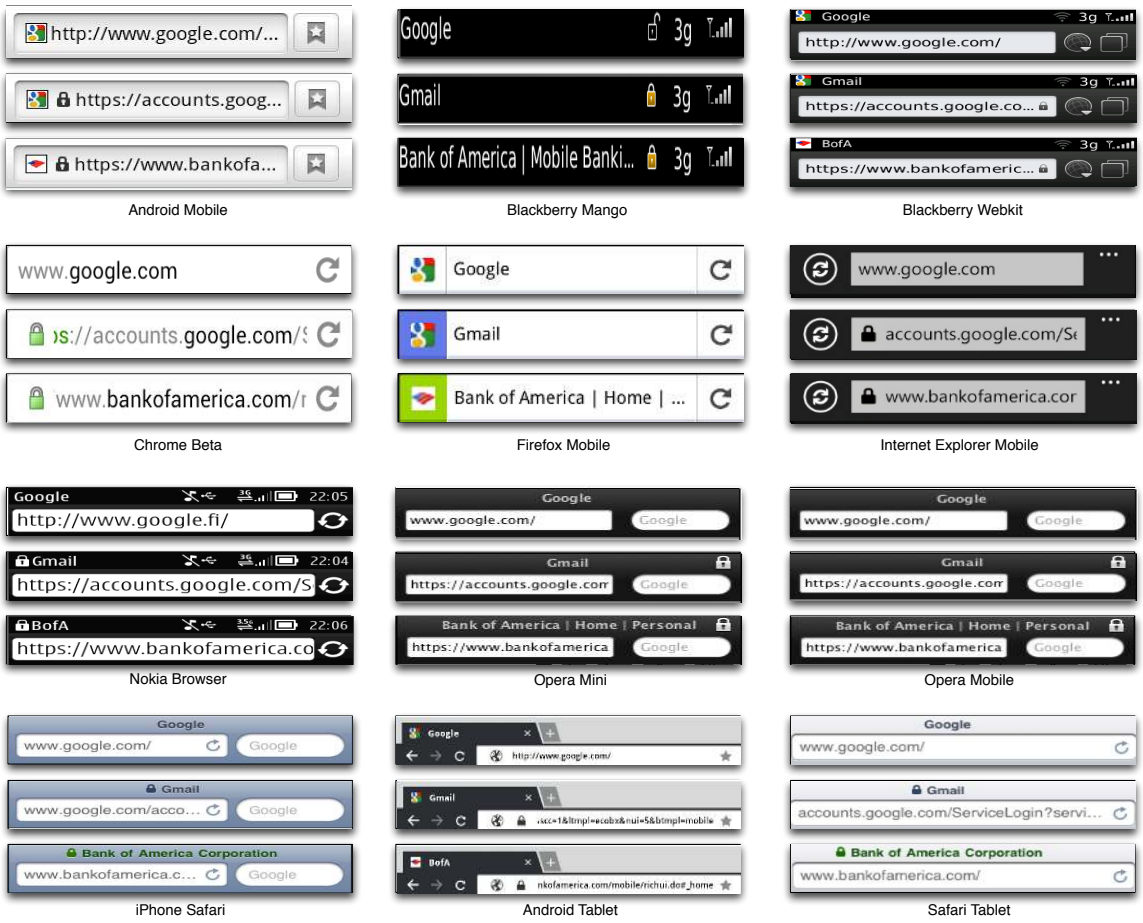

Figure 7: Security indicators on the primary interface (address bar) of all the mobile and tablet browsers. Every browser has three screenshots of the address bar: from top to bottom, the websites are Google over an `http` connection, Gmail over a secure connection with an SSL certificate and Bank of America over a secure connection with an EV-SSL certificate.

All the browsers display the `https` URL prefix either on the primary or the secondary interface. We note that this issue is present even in popular desktop browsers. The behavior of displaying the `https` URL prefix on a mixed content webpage fails to meet the W3C recommendation in both the desktop and mobile environments as shown in Tables 9 and 10.

b) Content and Indicator Proximity: The padlock icon used as a security indicator and the favicon used as an identity element of a website are two popular

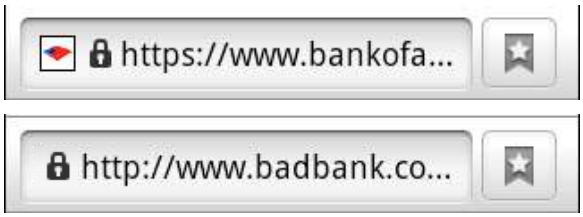

Figure 8: The address bar of the Android browser when a webpage over SSL is loaded. The browser places the favicon adjacent to the lock icon, thereby violating the W3C guideline 3b described in Section 4.2.2. The star icon to the right of the address bar is to bookmark the webpage.

elements that use a browser's chrome. If a browser allows a favicon to be placed next to the padlock, an attacker can feign a secure website by mimicking the favicon as a security indicator. We evaluate this scenario by visiting a webpage over a strong TLS connection from all candidate browsers and observed the relative locations of the favicon and padlock.

Observations: The Android mobile browser does not follow the W3C guideline. The browser places the favicon of a webpage beside the padlock icon as shown in Figure 8. All other browsers adhere to this guideline, as shown in Tables 9 and 10.

We observed several inconsistencies in the use and position of the padlock icon and the favicon in the mobile and tablet browsers. As shown in Figure 7, the favicon is displayed only on the Android (mobile and tablet), Blackberry Webkit and Firefox Mobile browsers. The remaining mobile and tablet browsers never display a favicon. This behavior is inconsistent with desktop browsers. We believe lack of screen space to be one of the drivers behind the removal of the favicon from the mobile environment. In addition to the almost total lack of use of favicons, we also noticed that the position of the padlock icon in mobile browsers is inconsistent across different mobile browsers. In the past, researchers have shown that the padlock icon is the security indicator most often noticed by users [100,211]. Traditional desktop browsers generally display the padlock icon in the address bar. However, all mobile and tablet

Table 11: Results of experiments on candidate mobile browsers to test compliance with the W3C guidelines 3c and 4 given in Section 4.2.2. The symbol notation is as defined in Table 7. 's': Implies that the `https` URL prefix is present on the 's'econdary interface.

Mobile and Tablet Browsers (See Table 6 for versions)	**TLS indicators**			**4) Robustness**
	3c) availability			Content obscures indicators on the address bar?
	`https` prefix available?	Lock shown?	Cipher details available?	
Android	•(s)	•	×	×
Blackberry Mango	•(s)	•	•	NA
Blackberry Webkit	•(s)	•	•	×
Chrome Beta	•	•	•	×
Firefox Mobile	•(s)	• On clicking the site identity button	×	×
iPhone Safari	•(s)	•	×	×
Nokia Browser	•(s)	•	×	×
Opera Mini	•(s)	•	×	×
Opera Mobile	•(s)	•	×	×
Windows IE Mobile	•(s)	•	×	×
Safari on iPad 2	•(s)	•	×	•
Android on Galaxy	•(s)	•	×	×

browsers except Android (mobile and tablet), Blackberry Webkit, Chrome Beta, and IE Mobile browsers display the lock icon on the title bar instead of the address bar. We believe that the reason behind this shift of location of the padlock icon in the mobile and tablet browsers is the non-persistent availability of the address bar to the user. Whenever a user starts interacting with a webpage, most mobile browsers hide the address bar to accommodate more content on the small screen.

c) Availability: We studied the presence of the lock icon, the `https` URL prefix and details of the cipher used in a TLS connection by visiting a TLS protected webpage using all candidate browsers. The padlock icon and the `https` URL prefix are

Table 12: Results of experiments on traditional web browsers to test compliance with the same guidelines as Table 9. The symbol notation is as defined in Table 7. 'p': Implies that the `https` URL prefix is present on the 'p'rimary interface.

Desktop Browsers (See Table 6 for versions)	**TLS indicators**			**4) Robustness**
	3c) availability			Content obscures indicators on the address bar?
	`https` prefix available?	Lock shown?	Cipher details available?	
Chrome	• (p)	•	•	•
Firefox	• (p)	• On clicking the site identity button	•	•
IE	• (p)	•	×	•
Opera	• (p)	•	•	•
Safari	• (p)	•	×	•

primary interface indicators and cipher information is a secondary interface indicator on desktop browsers.

Observations: Websites handling sensitive digital transactions (such as banks) ask users to search for the `https` URL prefix to ensure security of their transactions. Therefore, easy access to the `https` URL prefix is important. This indicator is present in the address bar (primary interface) of desktop browsers and is clearly visible to the user at all times. Among the mobile and tablet browsers, all but the Blackberry Mango browser display the `https` URL prefix in the address bar. The Blackberry Mango browser does not have an address bar and provides a choice to view the webpage's URL from the browser's options. This setting requires a user to be knowledgable of the change to be able to find the URL of the current webpage and also makes the `https` URL prefix a secondary interface indicator. Although the other mobile browsers display the `https` URL prefix in the address bar, they hide the address bar (except Chrome Beta) for better usability. In the Chrome Beta browser, if the URL of a webpage is longer than the screen size, the https URL prefix is hidden. Since a user is required to interact with the address bar to view the URL prefix of a webpage, the https URL prefix becomes a secondary interface indicator in all mobile

and tablet browsers. This increases the likelihood of a successful downgrade attack (e.g., SSLstrip [30] attack) on the mobile and tablet browsers, since a user requires effort to view the https URL prefix.

The information about the ciphers used in setting up the TLS connection between a website server and the user's browser is not available in any of the browsers except Blackberry Mango and Webkit. Accordingly, all the mobile and tablet browsers except two do not comply this W3C guideline for our experiments. Tables 11 and 12 provide the summary of our results.

4.3.4 Robustness: Visibility of Indicators

The TLS indicators generally found on the primary interface are lock icon, `https` URL prefix, URL coloring and site identity button. Typically, the address bar in a web browser holds these indicators. Therefore, we examined whether web content overwrites or pushes the address bar containing security indicators out of a user's view during browsing.

Observations: Presumably, in order to free up screen real estate for other purposes, the address bar on all but one mobile browser is overwritten by web content once a webpage is rendered and/or when a user starts interacting with the page. The IE Mobile browser always displays the address bar, when the user accesses content in the portrait view. However, the address bar is never displayed in IE Mobile when a user interacts with a webpage in the landscape mode. The Chrome beta browser makes the address bar persistently available in both the portrait and landscape modes. Out of the two tablet browsers, only the tablet Safari browser avoids the security indicators on the address bar being overwritten by a webpage's content, therefore allowing a persistent view of the security indicators on the primary interface. The Android tablet browser hides the address bar once a webpage is rendered. Tables 11 and 12

show that all the candidate desktop browsers follow this guideline unlike the mobile and tablet browsers.

4.3.5 Error Messages

We created example scenarios that demand the warning/caution and danger messages, given the definitions in the W3C document. The W3C document provides examples of scenarios that demand a danger alert. However, as the document does not specify any scenarios that should trigger warnings, we carried out our tests using the following scenario.

We classified the scenario of a browser rendering a mixed content webpage as one that should trigger a warning. This is because on a webpage with both insecure and secure content, the user may or may not interact with the insecure content on the webpage. Therefore, the browser system is unable to positively determine whether the user is at risk. In contrast, we used an example scenario given in the W3C document for our experiments on danger messages. The W3C document defines 'rendering a webpage presenting a self-signed certificate' as one that should trigger a danger message, since the certificate is not from a trusted root.

a) Interruption: We examined whether the mobile and tablet browsers display a warning or danger message in our test scenarios. We further observed the nature of the messages to confirm that they actually interrupt the user's actions as specified by the W3C guidelines and are not displayed at a position on the screen which a user can ignore and continue interacting with the website.

Observations: Only four mobile and tablet browsers (Android Galaxy, Blackberry Webkit, IE Mobile and Nokia) display a warning notifying the user of the existence

Table 13: Results of experiments on traditional web browsers to test compliance with the W3C guidelines 5a, 5b and 5c given in Section 4.2.2. The symbol notation is as defined in Table 7. *NA*: Implies that the concerned experiment is not applicable to that browser, the reasoning can be found in the text. (*: Our view is that a browser should display a warning message for a webpage holding mixed content, to avoid misleading users trained to interpret SSL indicators to mean that the (entire) webpage is secured.) $\times^*$: Implies that the browser fails to warn a user according to our view.

Mobile and Tablet Browsers (See Table 6 for versions)	**5) Error messages**			
	5a) Interruption		**5b) Proceeding options** (for warnings)	**5c) Inhibit Interaction** (for danger messages)
	Warning (mixed content)	Danger (self-signed cert)		
Android	$\times^*$	•	NA^*	•
Blackberry Mango	$\times^*$	•	NA^*	•
Blackberry Webkit	•	•	"Continue, Close connection, View cert, Trust cert" options	•
Chrome Beta	$\times^*$	•	NA^*	•
Firefox Mobile	$\times^*$	•	NA^*	•
iPhone Safari	$\times^*$	•	NA^*	•
Nokia Browser	•	•	•	•
Opera Mini	$\times^*$	$\times$	NA^*	$\times$
Opera Mobile	$\times^*$	•	NA^*	•
Windows IE Mobile	•	•	"Yes and No" options	•
Safari on iPad 2	$\times^*$	•	NA^*	•
Android on Galaxy	•	•	"Continue, View Certificate, Go Back" options	•

Table 14: Results of experiments on traditional web browsers to test compliance with the same guidelines as Table 11. The symbol notation is as defined in Table 7 and Table 11.

Desktop Browsers (See Table 6 for versions)	**5) Error messages**			
	5a) Interruption		**5b) Proceeding options** (for warnings)	**5c) Inhibit interaction** (for danger messages)
	Warning (mixed content)	Danger (self-signed cert)		
Chrome	$\times^*$ site identity button shows a warning	•	NA^*	•
Firefox	$\times^*$	•	NA^*	•
IE	•	•	"Yes, No" More info options	•
Opera	$\times^*$	•	NA^*	•
Safari	$\times^*$	•	NA^*	•

of insecure content on a mixed content webpage, before the webpage is rendered. The other browsers do not interrupt the user by displaying a warning. The iPhone

Safari browser shows a mixed content warning on a console that needs to be enabled by a user and is intended for developers. We believe that most iPhone Safari users are unlikely to enable the debug console, carefully browse the debug messages and therefore understand the presence of mixed content. Among the desktop browsers, only IE displays a mixed content warning, thereby interrupting a user.

The mobile and tablet browsers comply with the interruption guideline by displaying a danger message, when a webpage with a self-signed certificate is rendered. The Opera Mini browser is the only browser that does not display a danger message in this scenario; it simply renders the webpage and does not show any TLS indicators.

b) Proceeding options: We examined whether the warning message displayed for a mixed content webpage provides a user with more than one option to proceed after interruption.

Observations: Only the Android Galaxy, Blackberry Webkit, IE Mobile and Nokia browsers display a warning message when navigated to a mixed content webpage. The IE Mobile browser informs the user about the presence of unsecured content on the webpage and provides two options for continuing: <Yes, No>. However, there is no option to the user to view the certificate provided by the top-level website using a secured connection. Conversely, the Android Galaxy and Nokia browsers provide an option to view a website's certificate. The options presented by the Android Galaxy browser are <Continue, View Certificate, Go back> and those presented by the Nokia browser are <Options, Back>. The "Go back" and "Back" options navigate the user to a webpage viewed right before the mixed content webpage. The options provided by the Nokia browser are <Accept this time only, Accept permanently, Certificate details>. The Blackberry Webkit browser provides the options to <Continue, Close Connection (default), View Certificate, Trust Certificate>. Among

Figure 9: Danger message on iPhone Safari when a website presenting a self-signed certificate is accessed. This message interrupts the user and also inhibits the user from proceeding without interacting with the danger message first. Note that the website's URL has been anonymized for submission.

the desktop browsers, IE provides <Yes, No, More info> options to proceed when a mixed content webpage is rendered.

c) Inhibit interaction: This guideline requires a browser to prevent a user from interacting with a website that triggers a danger message, before user interaction with the danger message. We visited a website presenting an untrusted self-signed certificate from all the browsers.

Observations: All mobile and tablet browsers except Opera Mini display a danger message on receiving a self-signed certificate. Additionally, they restrict a user's interaction to the danger message. A user is unable to access the website content before explicitly interacting with the danger message. Figure 9 shows the danger message presented by the iPhone Safari browser before loading a webpage with a self-signed certificate. The Opera Mini browser does not show an error for self-signed certificates. It simply routes the user to the webpage presenting the self-signed certificate. All desktop browsers correctly follow this guideline. Table 11 and Table 12

summarize our experimental results of the error message guidelines.

4.4 User Deception and Potential Attacks

Table 15: Summary of potential attacks on candidate mobile browsers. A × implies that the attack is possible. A · implies that the corresponding attack is not possible on the browser.

Attacks	Android	Blackberry Mango	Blackberry WebKit	Chrome Beta	Firefox Mobile	iPhone Safari	Nokia Browser	Opera Mini	Opera Mobile	IE Mobile	Safari on iPad	Android on Galaxy
Phishing without SSL	×	•	•	•	•	•	•	•	•	•	•	•
Phishing with SSL	•	×	•	•	•	×	•	×	×	×	×	•
Phishing using a compromised CA	•	×	•	•	•	×	•	×	×	×	×	•
Industrial espionage/ Eavesdropping	×	×	×	×	×	×	×	×	×	×	×	×

The W3C user interface guidelines, which we use as a proxy for best practice, are an effort to communicate security information to users such that they can make informed decisions about websites that they visit. If these guidelines are not implemented by a browser, users are more easily misled about the identity of a website or the security of a connection. We discuss four attacks that are enabled on browsers violating one or more of the W3C guidelines. Table 15 provides a summary of potential attacks described in this section on the candidate browsers.

4.4.1 Deception Methods

We discuss techniques to confuse user perception about the security of a website's pages when rendered in a browser that fails to meet one or more of the W3C guidelines. A malicious website or a network attacker successful in deceiving a user is more likely to succeed in stealing a user's sensitive information (i.e., phishing). We assume that a malicious website principal or a network attacker has knowledge of the incomplete security indicators in popular mobile browsers. We also assume that an honest user visits a website using a mobile browser that does not follow one or more of the W3C guidelines discussed in Section 4.2.2. Violation of a single W3C guideline may permit

multiple attack vectors. The goal of an adversary is to trick the user into believing that they are interacting with a secure website when they are actually interacting with an insecure one.

1. **Close imitation of identity information:** A malicious website can closely imitate the identity of a legitimate website to confuse a user. Attackers often buy domains that very closely resemble a legitimate website's domain in addition to imitating the content of the legitimate website. For example, an attacker can buy the domain "bankofamericaa.com" that closely resembles the "bankofamerica.com" domain (the malicious one has an additional 'a' at the end of bankofamerica). Difficulty in clearly viewing a website's URL due to the constrained screen size of a mobile device allows for the possibility of a user overlooking the slight difference in the domain name. Additionally, the attacker can also obtain an inexpensive SSL certificate for his malicious website so that the browser shows SSL indicators to the user to confuse the user into believing that the malicious website is the legitimate one.

   An expert user can view a website's identity, domain name and reason behind trusting the website, presented in the website's certificate to identify the true owner of the phishing website and avoid divulging sensitive information. Absence of the identity information of a website, domain name and the reason of trusting that particular website certificate is a violation of guideline 1 and 2 given in Section 4.2.2.

2. **Lock icon spoofing:** The padlock icon is an important TLS indicator on the primary interface of a browser. The padlock combined with the presence of the `https` prefix in the URL signifies the presence or absence of SSL. The placement of the padlock icon is critical because it provides a clean indication of encryption and therefore security. If browsers situate a site's favicon next to the padlock

icon in the primary interface, the utility of the padlock icon is diminished.

A malicious website can make its favicon appear exactly the same as a user's browser's padlock icon. This provides the illusion of strong TLS encryption and allows an attacker to convince an expert user that his personal information (e.g., credit card number) is kept confidential and encrypted in transit. Moreover, the fake padlock makes the website appear more legitimate without the attacker purchasing any SSL certificates. Finally, a browser allowing an attacker to manipulate the contents of the security indicators with website content is a direct violation of guideline 3b described in Section 4.2.2.

3. **Cipher downgrade:** A man-in-the-middle (network attacker) can tamper with the initial messages sent by a client browser to establish an SSL connection with a website server and force the negotiation of a weak cipher. An expert user can refrain from providing sensitive information on a connection established using a weak cipher, by viewing the cipher negotiated for the TLS connection with the server. However, the same expert would be unable to detect the presence of a weak cipher on an SSL connection carrying sensitive data if the browser does not display cipher information of the connection. Such a browser fails to meet the W3C requirement 3c in Section 4.2.2.

4. **Substitute `http` for `https`:** The `https` URL prefix is a TLS security indicator signifying encryption on the channel between the client and the server. Since the `https` URL prefix is available on the primary user interface (address bar), a user can easily view this indicator while browsing sensitive data. If a network attacker changes the intended `https` website to an `http` website, a user may be able to recognize that he is using an unsecured connection by noticing the change from `https` to `http` in the address bar.

If the `https` prefix is not available to a user persistently, requirement 3c described in Section 4.2.2 is not followed.

5. **Mixed content:** The content of a webpage is interpreted as mixed if the top-level resource was retrieved through a strongly TLS protected `http` transaction and some dependent resources were retrieved through a weakly protected or unprotected `http` transaction. Dependent resources include inline images, stylesheets, script content, and frame content. If a browser displays security indicators defined for strongly TLS secured webpages on a webpage hosting mixed content, users gain a false notion of security for the unsecured content. This can lead to users providing sensitive information such as passwords while interacting with unsecured `http` content embedded in a webpage whose top resource is acquired over a strongly protected TLS connection. Moreover, this browser behavior does not comply with the W3C guideline 3a given in Section 4.2.2.

4.4.2 Attacks

An attacker can exploit one or more of the above techniques for user deception to launch a range of attacks. We describe four attacks in order of increasing effort required from an attacker.

i) Phishing without SSL: An attacker masquerades as a trustworthy entity in a phishing attack. By closely imitating a legitimate website's identity information in combination with lock icon spoofing, a malicious website can launch a phishing attack without SSL on a browser violating the W3C guidelines 1, 2 and 3b as follows.

An attacker buys a domain name that closely resembles the domain name of the legitimate website. For example, to spoof `www.bankofamerica.com`, the attacker buys the domain name `www.bankofamericaa.com`. The attacker then imitates the content of the targeted legitimate website. Instead of spending money on purchasing an SSL certificate to increase the "false" credibility of the malicious website, an attacker

instead makes the favicon of the malicious website a lock image. Therefore, the closely imitated domain name provides an impression of correct identity of the intended website and the spoofed lock provides an illusion of strong encryption.

When this malicious website is rendered in a browser that makes viewing the URL of the website difficult, situates the favicon next to the padlock icon and does not offer a UI to view identity information such as website owner's name, even an advanced user might be subjected to phishing.

ii) Phishing with SSL: Spoofing only the lock icon may not be adequate to launch a successful phishing attack. To increase the credibility of a phishing website, the attacker can buy an inexpensive SSL certificate for the website. The presence of a valid certificate causes a browser to display SSL indicators such as the `https` URL prefix and URL coloring (or colored site identity button) in addition to the lock icon in the browser's chrome. If a user blindly trusts just these SSL indicators and can not verify additional identity information of the website (violation of guideline 1 and 2), he can be subjected to a phishing attack.

iii) Phishing using a compromised CA: Compromising a CA allows an attacker to obtain rogue certificates for legitimate websites. There have been several such attacks recently [37, 38]. If a user's browser trusts a CA, the browser will accept all certificates signed by the CA without showing any warning to the user. This behavior persists even when the same CA is compromised and the necessary update to remove the trusted CA from the browser has not been installed. An expert user who is knowledgeable of a CA compromise can verify every certificate issuer's organization in the certificate chain, therefore declining interacting with a malicious website with a rogue certificate. If a browser fails to meet guidelines 1 and 2, thereby not presenting user interface to enable certificate viewing, even an expert user could be exposed to a phishing attack.

iv) Industrial espionage / eavesdropping: A man-in-the-middle (network) attacker can use any one of the cipher downgrade, substituting `http` for `https` or inserting mixed content techniques for user deception to launch an eavesdropping attack on a user's session as follows:

SSLstrip attack: The SSLstrip [30] man-in-the-middle attacker sits on a local network and intercepts traffic. When the attacker detects a request to an encrypted `https` site, he substitutes a duplicate of the intended destination as an unencrypted `http` site. This switching strips away the security that prevents a third party from stealing or modifying data, while deceiving the server that an encrypted page has been sent to the client. The network attacker can also fake a lock icon in the stripped `http` page, by replacing the favicon by a lock icon [154]. If the `https` prefix is not available to a user persistently, he may not be able to recognize that he is using an unsecured connection by noticing the change from `https` to `http` in the address bar. A browser not displaying the `https` prefix persistently does not follow requirement 3c in Section 4.2.

Cipher downgrade attack: A man-in-the-middle (network attacker) can tamper with the initial messages sent by a client browser to establish an SSL connection with a website server. Before a TLS connection is set up, a client and server exchange a list of ciphers that they support. A network attacker can modify the list of supported ciphers sent by the client to a list containing only weak ciphers, and then forward the client's request/response to the server. On receiving a list of only weak ciphers (e.g., DES-CBC-SHA), the server can either drop the connection because no ciphers are mutually supported, or provide support for that cipher and begin an encrypted session with the weak cipher. When a connection using the weak cipher is initiated, all the data in transit is protected using the weak cipher's encryption scheme. This allows a network attacker to capture the stream of data and break the weak encryption offline. The attack is also useful to mislead even an expert user that their transactions are

over a connection with strong encryption algorithms, since the SSL indicators such as `https` URL prefix and lock icon are present even for a connection using a weak cipher. If a browser does not display cipher information, it fails to meet the W3C requirement 3c in Section 4.2.

Mixed content attack: A man-in-the-middle attacker can tamper (e.g., code injection) with the unencrypted content present on a webpage consisting of mixed content and replace the original content with any malicious content of his choice. If a web browser displays SSL indicators for a webpage containing mixed content (violation of guideline 3a), even an expert user may be unable to detect a network attack exploiting the mixed content on a webpage.

Our experimental results combined with this threat model make the candidate mobile and tablet browsers susceptible to phishing and eavesdropping attacks as shown in Table 15.

4.5 Additional results

We discuss our findings that are not directly related to the guidelines studied in Section 4.3. We note our observations on the positive and negative characteristics shown by the mobile and tablet browsers. We also discuss an important security scenario that is not represented in the W3C guidelines and argue that it requires attention. Table 16 provides a summary of the results covered in this section.

4.5.1 The Good

The W3C document defines two guidelines that MUST hold when strong TLS algorithms are negotiated between a client and a server:

1. No version of the TLS protocol that suffers known security flaws has been negotiated. At the point of writing of this document, versions of SSL prior to SSLv3 MUST NOT be considered strong.

Table 16: Results of the support for SSLv2, the null cipher, DES-CBC-SHA (weak cipher) and whether browsers differentiate between EV-SSL and SSL certified webpages. The symbol notation is as defined in Table 7.

Mobile and Tablet Browsers (See Table 6 for versions)	SSLv2 supported?	Null cipher supported?	Weak cipher prohibited? (DES-CBC-SHA)	EV-SSL vs SSL differentiation?
Android	•	•	×	×
Blackberry Mango	•	•	×	×
Blackberry Webkit	•	•	×	×
Chrome Beta	•	•	•	×
Firefox Mobile	•	•	•	• (site identity button coloring)
iPhone Safari	•	•	×	• (title URL coloring)
Nokia Browser	•	•	×	×
Opera Mini	•	•	•	×
Opera Mobile	•	•	•	×
Windows IE Mobile	•	•	•	×
Safari on iPad 2	•	•	×	• (title URL coloring)
Android on Galaxy	•	•	•	×

2. A cipher suite has been selected for which key and algorithm strengths correspond to industry practice. The "export" cipher suites explicitly prohibited in appendix A.5 of TLSv11 [26] (RFC 4346) MUST NOT be considered strong.

To verify the compliance with these guidelines we conducted two experiments.

SSLv2: We browsed to a website supporting only SSLv2 from each of the candidate browsers. We found that all the mobile, tablet and desktop browsers comply with the first guideline and do not support SSLv2.

Null cipher: The null cipher is one of the prohibited ciphers in RFC 4346 and one of the most dangerous ciphers because it represents the lack of an encrypted communication channel. To test browser compliance with the second guideline for strong TLS algorithms, we built a website that supports only the null cipher. We

observed that none of the mobile, tablet or desktop candidate browsers support the null cipher.[2]

Discontinuing support for SSLv2 and the null cipher automatically reduces the probability of cipher downgrade attacks on the candidate browsers.

4.5.2 The Bad

A browser supporting a weak cipher can enable a network attacker to break the encrypted messages offline. The SSLv3 cipher-suite consists of certain weak ciphers, although they are stronger than the SSLv2 ciphers and the null cipher. We verified the support of the DES-CBC-SHA weak cipher. We observed that six (Android Mobile, Blackberry Mango and Webkit, iPhone and iPad2 Safari and Nokia Browser) out of the eleven mobile and tablet browsers support the weak cipher. The other mobile and tablet browsers display error messages conveying the absence of a common encryption protocol with the server. It is interesting to note that the the Safari browser in its mobile, tablet and even desktop versions supports this weak cipher. However, the Android tablet browser does not support this cipher, unlike its mobile version. Since most mobile and tablet browsers do not allow users to see the cipher used on a TLS connection, they can not determine that a weak cipher is being used. No desktop browser other than Safari support this cipher.

4.5.3 The Silent

The W3C document does not establish guidelines for the browser user interface to signify the difference between EV-SSL [34, 192] and SSL certificates. The sole distinction between an SSL and an EV-SSL certificate from a user's perspective is the set of indicators on his browser. For example, the Firefox desktop browser uses a green site identity button to convey the presence of an EV-SSL certificate on a website.

[2]We did not test for the support to all the prohibited ciphers (as given in TLSv11 [26]) by the candidate browsers.

However, the site identity button is blue in the same browser when a website with an SSL certificate is rendered.

SSL certificates can be 'domain-validation-only' with minimal verification performed on the details of the certificate. Since any successful SSL connection causes the padlock icon to appear, users are not likely to be aware of whether the website owner has been validated or not. Therefore, fraudulent websites have started using inexpensive domain-validated SSL certificates with minimal verification to gain user trust. EV-SSL certificates were created to restore confidence among users that a particular website has been subjected to more rigorous vetting and has a verifiable identity. If browsers do not differentiate between SSL and EV-SSL certificates, then the fundamental motivation [34] behind EV-SSL certificates becomes void, so too does the incentive for site owners to pay extra for such certificates. An SSL certificate from Go Daddy costs $12.99/year [9] and an EV-SSL certificate from VeriSign costs $1499/year [22]. In a browser with no differentiation between SSL and EV-SSL, both these certificates are the same from a user's perspective. An adversary holding a domain name and willing to spend money for the SSL certificate would then trigger exactly the same user interface elements to users, and thus appear to provide identical guarantees as a website certified by the more expensive certificate.

Experimental observations: We browsed both EV-SSL and SSL certified webpages using all the candidate browsers. With the exceptions of the Firefox Mobile and the iPhone and iPad Safari browsers, none of the mobile or tablet browsers display any indicators that differentiate between EV-SSL and SSL certified webpages. The Firefox Mobile browser uses green and blue colors of the site identity button to depict the presence of EV-SSL and SSL certified webpages respectively. The Safari mobile and tablet browsers use green and blue coloring of the 'title' to represent the difference between EV-SSL and SSL. This behavior of the Firefox Mobile and the Safari browsers

is consistent with their desktop counterparts. However, the IE Mobile, and the Opera Mini and Mobile browsers are not consistent with the methods used on their desktop counterparts to portray the different between EV-SSL and SSL webpages.

Gauging the security level of a website using the different EV-SSL and SSL indicators can be complicated for an average user. The inconsistency across the mobile and desktop browsers from the *same* vendor adds to an already confusing task. We believe that a clear guideline on the indicators for differentiating between EV-SSL and SSL certified webpages is necessary to help browser vendors provide the expected interface consistently in the desktop and mobile environments. Moreover, we suggest that a guideline from a well established international standards organization such as the W3C is a minimal starting point in order to achieve consistency across browser software from different vendors.

We note the following advice within official guidelines from the CA/Browser Forum [29]: *In cases where the relying application accepts both EV and non-EV certificates, it is recommended that the application's behavior differ in a distinct way for each type of certificate. Application developers should consider the EV treatment offered by other application developers that also recognize EV certificates and, where practical, provide consistent treatment.* We believe that much more specific advice is essential, for example, in a revision or extension of the W3C user interface guidelines [35].

4.6 Discussion and Concluding Remarks

4.6.1 Discussion

For this study, we selected a subset of the absolute requirements and prohibitions from the W3C guidelines. From our experimental analysis, we observed that popular mobile and tablet browsers fail to meet many of the guidelines. However, by and large popular desktop browsers follow the set of guidelines studied in this chapter. Furthermore, the inconsistencies that we observed herein are cause for significant

concern, as consistency of user interfaces is recognized as a fundamental usability attribute [132], conveying many user benefits including enhancing users' ability to transfer skills across similar systems, and reducing training time on new and related systems. Moreover, related to security, inconsistency confuses users, and confusion aids the attacker. Our results raise an important question: *is it appropriate to apply the same user interface guidelines for web security to both the desktop and the mobile environments?*

We believe that the non-conformance to the W3C guidelines and the inconsistencies in the use and presentation of SSL indicators in mobile browsers is primarily due to the adjustments made in the browser interface as a result of the tension between usability and security, and possibly due to dis-connects between mobile and desktop development teams. For example, the address bar consisting of the padlock icon and the `https` URL prefix indicators is persistently available in desktop browsers, however is hidden (apparently to better accommodate content on small mobile screens) for the majority of the time during user interaction. It is cumbersome for a user to bring the address bar in view (to observe indicators) by scrolling to the top of the mobile screen, suggesting this will be done far less frequently, whereas viewing the indicators on a desktop browser requires little or no extra effort. This ease of interaction with desktop browsers also makes consuming certificate information simpler for a user. Again in contrast, current design decisions related to mobile screen real estate force users to execute scrolling operations to view all the content of a certificate, implying greater inconvenience and effort compared to consuming certificate information on desktop browsers; in other cases, mobile browser vendors have decided to make certificate information entirely unavailable. Such significant design changes preclude even expert users from discerning clues about the credibility and security of websites, due to the absence of security indicators, leaving average users with no hope at all. These security concerns, the very significant non-conformance with existing recommendations,

and tremendous inconsistency both within and across browser vendors, lead us to call for the establishment of new recommendations for the mobile environment that specifically take into account its limitations and additional challenges.

4.6.2 Conclusion

Modern mobile browsers enable a range of sensitive operations over SSL/TLS connections. Although these browsers aim for equivalent functionality to traditional desktops, their smaller screen size has resulted in significant changes to the presentation and availability of SSL indicators. We have carried out the first evaluation of security indicators in mobile browsers, using the W3C web interface guidelines to measure compliance in ten mobile and two tablet browsers. We observed that mobile browsers fail to meet many of the security guidelines and exhibit tremendous inconsistency in the presentation and availability of SSL indicators in contrast to traditional desktop browsers. Such significant design changes preclude even expert users from discerning clues about the credibility and security of websites, raising significant concerns about the security of average users. Additionally, we observed that the absence of clear and consistent EV-SSL indications leads to EV-SSL certificates currently adding complexity to the mobile ecosystem without any corresponding benefits. Our work may be viewed as a call to arms for greater consistency in mobile browser security interfaces and greater attention to the specific challenges of mobile device issues in security user interface guidelines. It also raises questions about the utility and usability of security indicators as presently implemented in mobile browsers and related questions about the viability of extended validation SSL certificates in light of current mobile browser user interfaces.

CHAPTER V

KAYO: DETECTING MOBILE MALICIOUS WEBPAGES IN REAL-TIME

5.1 Introduction

Mobile devices are increasingly being used to access the web. However, in spite of significant advances in processor power and bandwidth, the browsing experience on mobile devices is considerably different. These differences can largely be attributed to the dramatic reduction of screen size, which impacts the content, functionality and layout of mobile webpages.

Content, functionality and layout have regularly been used to perform static analysis to determine maliciousness in the desktop space [84, 147, 176]. Features such as the frequency of iframes and the number of redirections have previously served as strong indicators of malicious intent. Due to the significant changes made to accommodate mobile devices, such assertions may no longer be true. For example, whereas such behavior would be flagged as suspicious in the desktop setting, many popular benign mobile webpages require multiple redirections before users gain access to content. Previous techniques also fail to consider mobile specific webpage elements such as calls to mobile APIs. For instance, links that spawn the phone's dialer (and the reputation of the number itself) can provide strong evidence of the intent of the page. New tools are therefore necessary to identify malicious pages in the mobile web.

In this chapter, we present kAYO[1], a fast and reliable static analysis technique to detect malicious *mobile* webpages. kAYO uses static features of mobile webpages

[1]Our technique is called "kAYO" (knockout in boxing terminology) because it knocks out malicious mobile webpages.

derived from their HTML and JavaScript content, URL and advanced mobile specific capabilities. We first experimentally demonstrate that the distributions of identical static features when extracted from desktop and mobile webpages vary dramatically. We then collect over 350,000 mobile benign and malicious webpages over a period of three months. We then use a binomial classification technique to develop a model for kAYO to provide 90% accuracy and 89% true positive rate. kAYO's performance matches or exceeds that of existing static techniques used in the desktop space. kAYO also detects a number of malicious mobile webpages not precisely detected by existing techniques such as VirusTotal and Google Safe Browsing. Finally, we discuss the limitations of existing tools to detect mobile malicious webpages and build a browser extension based on kAYO that provides real-time feedback to mobile browser users.

We make the following contributions:

- **Experimentally demonstrate the differences in the "security features" of desktop and mobile webpages:** We experimentally demonstrate that the distributions of static security features used in existing techniques (e.g., the number of redirections) are different when measured on mobile and desktop webpages. Moreover, we illustrate that certain static features are inversely correlated to a webpage being malicious, when extracted from desktop and mobile pages. The results of our experiments demonstrate the need for mobile specific techniques for detecting malicious webpages.

- **Design and implement a classifier for malicious and benign mobile webpages:** We collect over 350,000 benign and malicious mobile webpages. We then identify a range of new mobile relevant static features from these webpages that distinguish between mobile benign and malicious webpages. kAYO provides 90% accuracy in classification and shows improvement of two orders of magnitude in the speed of feature extraction over similar existing techniques.

We further empirically demonstrate the significance of kAYO's features. Finally, we also identify 173 mobile webpages implementing cross-channel attacks, which attempt to induce mobile users to call numbers associated with known fraud campaigns.

- **Implement a browser extension based on kAYO:** To the best of our knowledge kAYO is the first technique that detects mobile specific malicious webpages by static analysis. Existing tools such as Google Safe Browsing are not enabled on the mobile versions of browsers, thereby precluding mobile users. Moreover, the mobile specific design of kAYO enables detection of malicious mobile webpages missed by existing techniques. Finally, our survey of existing extensions on Firefox desktop browser suggests that there is a paucity of tools that help users identify mobile malicious webpages. To fill this void and for immediate use, we build a Firefox mobile browser extension using kAYO, which informs users about the maliciousness of the webpages they intend to visit in real-time.

The remainder of the chapter is organized as follows: Section 2 provides an overview of related research. Section 3 experimentally motivates the need for mobile specific static techniques. The static feature set used in kAYO followed by the collection process of the data used in modeling kAYO is described in Section 4. Section 5 provides details about the machine learning techniques used, the implementation and evaluation of kAYO's model, the effectiveness of kAYO's features, and comparison of kAYO with existing techniques. The browser extension architecture is described in Section 6. Section 7 presents case studies of malicious mobile webpages detected by kAYO in the wild and also discusses the limitations of our technique and future work. Section 8 provides concluding remarks.

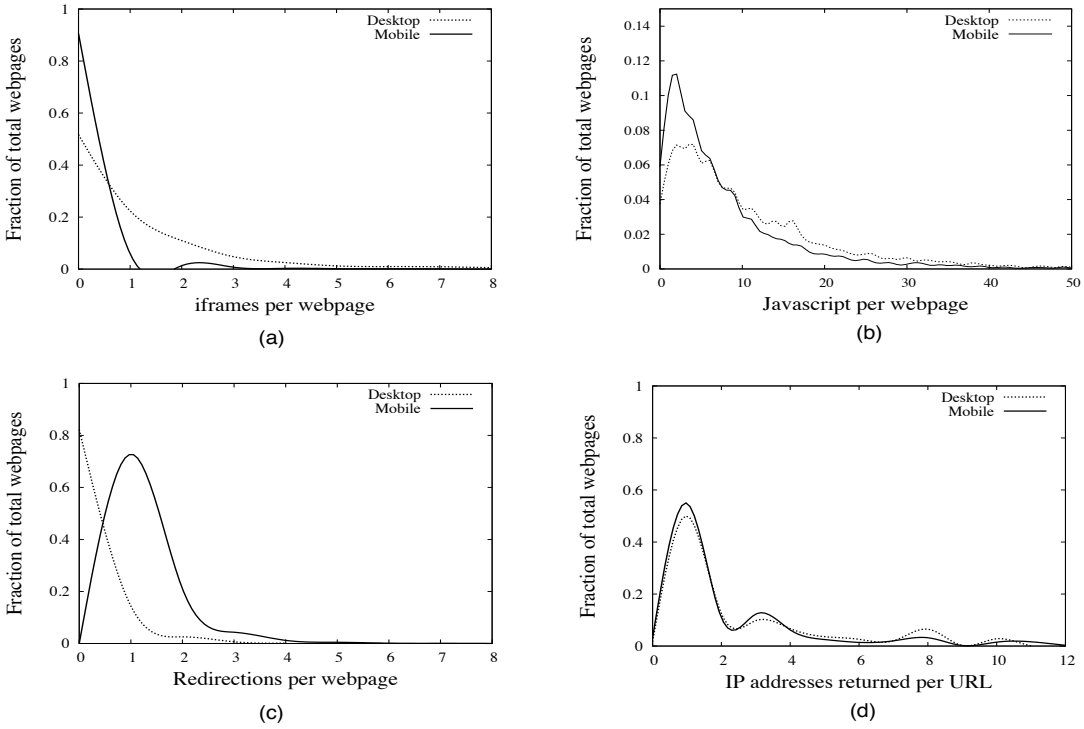

Figure 10: Normalized density curves of static features when measured on mobile and desktop webpages. There is substantial difference between the distributions of the number of (a) iframes, (b) Javascript and (c) redirections when measured on mobile and desktop versions of the same websites, whereas, the distribution of the number of (d) IP addresses is similar.

5.2 Motivation

Static analysis techniques to detect malicious websites often use features of a webpage such as HTML, JavaScript and characteristics of the URL. Usually, these features are fed to machine learning techniques to classify benign and malicious webpages. These techniques are predicated on the assumption that the features are distributed differently across benign and malicious webpages. Accordingly, any changes in the distribution of static features in benign and/or malicious webpages impacts classification results of static analysis techniques. While successful, these static analysis techniques have been used *exclusively* for desktop webpages [84, 176, 218].

Mobile websites are significantly different from their desktop counterparts in content, functionality and layout. Consequently, existing tools using static features to detect malicious desktop webpages are unlikely to work for mobile webpages. We explain four factors that motivate building separate static analysis techniques to detect malicious mobile webpages.

1) Differences in content: Mobile websites are often simpler than their desktop counterparts. Therefore, the distribution of content-based static features (such as the number of JavaScripts) on mobile webpages differs from that of desktop webpages. For example, Figure 10 (a) and Figure 10 (b) show the normalized density of the number of iframes and the number of Javascript found in mobile[2] and the corresponding desktop versions of the top-level webpage of the 10,000 most popular websites from Alexa [46]. Approximately 90% of mobile webpages do not have any iframes, whereas the corresponding desktop webpages have multiple iframes. Desktop webpages have more Javascripts than mobile webpages.

Due to the simplicity of mobile webpages, the majority of other content related static features used in existing techniques including, the number of images, page length, the number of hidden elements, and the number of elements with a small area also differ in magnitude in mobile and desktop webpages.

2) Infrastructure: Website providers use JavaScript or user agent strings to identify and then redirect mobile users to a mobile specific version. Figure 10 (c) shows the normalized density of the number of redirection steps taken by the desktop and mobile versions of the top 10,000 websites on Alexa before landing on the final URL[3]. Even the most popular mobile websites show multiple redirects, which has traditionally been a property of desktop websites hosting malware [176]. However, multiple redirects does not necessarily indicate bad behavior for mobile websites due to the

[2]We describe the method used to define and identify mobile webpages in detail in Section 5.3.2.

[3]We use the term *final URL* to denote the URL that is rendered in the browser after redirections (if any) from the seed URL. The final URL may change based on the browser's user agent.

characteristics of their hosting infrastructure.

We note that not all static features used in existing techniques differ when measured on mobile and desktop webpages. For example, the number of IP addresses returned by DNS servers for mobile and desktop versions of the same sites are comparable. Mobile websites appear to share their hosting infrastructure with the corresponding desktop websites [146]. We used seven public (Google, OpenDNS, UltraDNS, Norton, DynDNS, Level3, and Scrubit) DNS servers to obtain the IP addresses returned in the DNS A records of mobile and corresponding desktop URLs of Alexa top 10,000 websites. As seen in Figure 10 (d), the distributions of the number of IP addresses returned by the seven DNS servers are similar for mobile and desktop websites.

3) Impact of screen size: The screen size of a mobile phone is significantly smaller that that of a desktop computer. Therefore, a mobile user only sees a part of the URL of a webpage. Intuitively, the author of a mobile phishing webpage may only need to include misleading words at the beginning of the URL and a short URL might suffice to trick a user.

4) Mobile specific functionality: Mobile websites enable access to a user's personal information and advanced capabilities of mobile devices through web APIs. Existing static analysis techniques do not consider these mobile specific functionalities in their feature set. We argue and later demonstrate that accounting for the mobile specific functionalities helps identify new threats specific to the mobile web. For example, the presence of a known 'bank' fraud number on a website might indicate that the webpage is a phishing webpage imitating the same bank [64].

Limitations of existing techniques: These discrepancies between mobile and desktop webpages demand investigation. Existing static analysis techniques and tools for detecting malicious webpages are focused on desktop webpages. *Therefore, they*

are unable to detect mobile specific threats with high accuracy.[4] Secondly, several webpages built specifically for mobile, return empty pages when rendered in a desktop browser. Thus, even existing dynamic analysis techniques that execute websites in desktop browsers on virtual machines, are ineffective on such mobile websites. Finally, signature based tools such as Google Safe Browsing currently only work with desktop browsers. We manually visited five mobile specific known malicious webpages collected from PhishTank [16], from the Google Chrome mobile browser. We observed that these webpages are flagged as malicious on the Chrome desktop browser, but not on the Chrome mobile browser whose users are the real targets of the mobile malicious webpages. Although enabling Google Safe Browsing in mobile Chrome is an engineering effort, we argue and later demonstrate that a mobile specific static technique can also detect new threats previously unseen by such services.

Goals: Considering the limitations of existing techniques, the goals of this work are three-fold. First, identifying relevant static features from mobile specific webpages in the wild. Second, implementing a fast and reliable static analysis technique to detect malicious mobile webpages in real-time. And finally, developing a mobile browser extension that will inspect mobile webpages in real-time and provide feedback to the user.

5.3 Methodology

Our objective is to design and develop a technique to identify mobile specific malicious webpages in real-time. We extract static features from a webpage and make predictions about its potential maliciousness. We first discuss the feature set used in kAYO followed by the collection process of the dataset.

[4]We demonstrate this experimentally in Section 5.4.

Table 17: The 44 features of kAYO from four categories. According to our knowledge, the category of mobile specific features is studied for the first time in this work. The significance of these features is described in Section 5.4.2.

Category	**Features**	**Total # of features**
Mobile specific	# of API calls to tel:, sms:, smsto:, mms:, mmsto:, geolocation; # of apk, # of ipa	8
JavaScript	presence of JS, noscript, internal JS, external JS, embedded JS; # of JS, noscript, internal JS, external JS, embedded JS	10
HTML	presence of internal links, external links, images; # of internal links, external links, images # of cookies from header, secure and HTTPOnly cookies, presence of redirections and iframes, # of redirects and iframes, whether webpage served over SSL, % of white spaces in the HTML content	14
URL	# of misleading words in the URL such as *bank*, # of forward slashes and question marks, digits, # of dots, hyphens and underscores, # of equal signs and ampersand, subdomains, # of two letter subdomains, semicolons, presence of subdomain, % of digits in hostname length of URL	12
	Total:	44

5.3.1 kAYO Feature Set

A webpage has several components including HTML and JavaScript code, image files, the URL, and header information. Additionally, mobile specific webpages also access applications running on a user's device using web APIs (e.g., the dialer). We extract structural, lexical and quantitative properties of such components to generate kAYO's feature set. We focus on extracting mobile relevant features that take minimal extraction time. Our hypothesis is that such features are strong indicators of whether a webpage has been built for assisting a user in their web browsing experience or for malicious purposes.

Our feature set consists of 44 features in total, 11 of which are new and not previously identified or used. We describe the newly identified features in detail. A subset of features in kAYO have been used by other authors in static inspection of

desktop webpages in the past.[5] However, it is important to note that these features when extracted from mobile webpages and desktop webpages differ in magnitude (e.g., number of iframes) and show varying correlation with the nature of the webpage (i.e., malicious/benign).

We divide kAYO's 44 features into four classes: mobile specific features, JavaScript features, HTML features and URL features. To the best of our knowledge, we are the first to use these mobile specific features. Table 17 summarizes the 8 mobile, 10 JavaScript, 14 HTML and 12 URL features. We empirically illustrate the effectiveness of each of the features in Section 5.4.2.

5.3.1.1 Mobile specific features

We collect eight mobile specific features to capture the advanced capabilities of mobile webpages. Mobile websites enable access to personal data from a user's phone, an experience not offered by desktop websites. For example, APIs such as `tel:` and `sms:` spawn the dialer and the SMS applications respectively on a mobile device. In order to characterize the behavior of mobile API calls, we extracted the number of API calls `tel:`, `sms:`, `smsto:`, `mms:` and `mmsto:` from each mobile webpage. We further extracted the target phone numbers from these API calls. We ran the commercially available Pindrop Security Phone Reputation System (PRS) [17] on each phone number. Based on the results of the PRS, we gave the score of 1/0 (known fraud/benign) to each phone number scraped from the mobile API calls, and added the score as a feature in kAYO. We only extracted phone numbers with API prefixes that could trigger an application installed on a user's phone. We did not consider phone number strings simply listed on webpages without an API prefix.

We argue that due to the popularity of application markets such as Google play

[5] A subset of kAYO's features was selected from prior literature on desktop webpage classification using static features. Only the features deemed to be the most critical and definitively applicable to mobile webpages as shown by manual analysis, were selected based on the authors' experience and knowledge of the area.

and iTunes, a website hosting its own mobile application binary (e.g., *.apk* or *.ipa* files), might suggest bad behavior. Therefore, we added number of *.apk* and *.ipa* files found on a webpage as a feature. If we found more than a threshold (in the few hundreds) of apk/ipa files on the same webpage, we assumed that the webpage was an app store and was unlikely to be malicious.

5.3.1.2 JavaScript features

JavaScript enables client-side user interaction, asynchronous communication with servers, and modification of the DOM objects of webpages on the fly. We extract 10 features that capture the JavaScript relevant static behavior of a webpage, two of which are new. All the features are faster to extract than the features based on JavaScript deobfuscation.

JavaScript found on malicious webpages can be obfuscated. Instead of deobfuscating every JavaScript, we extract simple JavaScript related features from a webpage. The primary reason in choosing this approach is that a large number of benign webpages include potentially dangerous JavaScript code as shown by Yue et al. [215]. For example, 44.4% of the top 6,805 websites from Alexa use the potentially dangerous `eval` function. These observations invalidate the assumption made in existing techniques [84]; that potentially dangerous JavaScript keywords are more frequently used in malicious webpages. Secondly, external JavaScript can be very large, sometimes of the order of a few megabytes. Our goal is to build a real-time browser extension based on kAYO. Accordingly, we avoided using features that would slow down the feature extraction process.

We argue that benign webpage writers take efforts in providing good user experience, whereas, the goal for malicious webpage authors is to trick a user into performing unintentional action with minimal effort. We therefore determine whether a webpage

has `noscript` content and also measure the number of `noscript`. Intuitively, a benign webpage writer will have more `noscript` in the code to ensure good experience even for a security savvy user. We add these two newly identified features to our set.

Webpages generally include three types of JavaScript: internal, external and embedded. An internal JavaScript is one hosted on the same domain as that of its parent webpage, whereas, an external JavaScript's domain is different from its host's domain. Both internal and external JavaScript are simply links to JavaScript hosted elsewhere. Since mobile webpages are often simpler than desktop webpages and phishing is the biggest threat on mobile webpages at present, we expect that benign webpages will include more external JavaScript for advertisements and analytics purposes, whereas malicious webpages will have a lower number of external JavaScript. Accordingly, we determine whether a webpage holds external and internal JavaScript, and then extract the number of internal and external JavaScript from a webpage. Unlike internal and external JavaScript, embedded JavaScript code is contained in the webpage. If the number of lines of JavaScript is relatively small, a webpage with embedded JavaScript loads faster than pages that must reference external code. This is because, as the web browser loads the page and encounters the reference to the external code, it must make a separate request to the web server to fetch the code. Webpages built for performance often use a number of embedded JavaScript. Performance is critical in the mobile web since it impacts revenue and user interest [174]. Therefore, we determine whether a webpage hosts embedded JavaScript and then calculate the number of embedded JavaScript in a webpage. Our assumption is that on average, benign webpages will have more embedded JavaScript. Finally, we determine whether JavaScript is present at all on a webpage, and measure the total number of JavaScript on the webpage including embedded, internal and external.

5.3.1.3 HTML features

We extract 14 features in total from the HTML code of each webpage. Popular webpages include a number of images, and internal and external HTML links for better user experience. For example, the top-level page of *m.cnn.com* includes links to other news articles published by CNN (internal HTML links), advertisements for a local restaurant (external HTML link) and images related to the latest breaking news. Accordingly, we first determine whether a webpage has any images, internal and external HTML links. We then extract the number of internal links, external links and images from a webpage as features of kAYO.

Malicious webpages (especially those implementing drive-by-downloads and click-jacking) include links to bad content in iframes [176]. Recall that the distribution of iframes on mobile webpages is different as compared to that on desktop webpages. However, we do not rule out the possibility of a mobile malicious webpage including malicious content in iframes and consider the presence and number of iframes in a webpage as features in kAYO. Past research also shows that malicious websites take several redirections before leading the user to the target webpage to avoid DNS based detection [176]. Recall that mobile webpages generally take at least one or more redirections since both desktop and mobile versions of the webpage share hosting infrastructure. Therefore, we determine whether a webpage was redirected and then measure the number of redirections the user experiences before landing on the final URL. Finally, we extract other features such as the percentage of white spaces in the HTML content, the number of cookies from the header, the number of secure and HTTPOnly cookies, and whether the webpage is served over an SSL connection. Readers are encouraged to refer to prior literature [84, 156, 218] for more information on the usefulness of these HTML features.

5.3.1.4 URL features

Structural and lexical properties of a URL have been used to differentiate between malicious and benign webpages. However, using only URL features for such differentiation leads to a high false positive rate. We extract 12 URL features in total.

Authors of phishing webpages often exploit the familiarity of users to a webpage [97] by including words in the URL that can mislead a user into believing that the phishing webpage is the legitimate webpage. Words such as *login* and *bank* are commonly used in the URL of the login webpage for benign websites that are highly prone to imitation. Only a part of the URL is visible to the user of a mobile phone due to the small screen [58]. Therefore, intuitively, the author of a phishing webpage will include misleading words at the beginning of the URL. We consider the presence of such words in the URL as a new feature in kAYO.

A significant number of phishing domain names are simply IP addresses of machines hosting them [114, 155]. Therefore, we calculated the number of digits in a URL and the percentage of digits in the hostname. Phishing webpage developers usually create a number of subdomains to include deceptive keywords such as `paypal` as a subdomain. This might increase the length of phishing URLs [155]. Therefore, we include the length of a URL, whether the URL contains a subdomain, the number of subdomains, and the number of dots as features. Our URL feature set also contains the number of semicolons, equal signs and ampersand symbols, hyphens and underscores, forward slashes and question marks. Interested readers are referred to prior literature [114, 138, 153] for details on the importance of these URL features.

Note that the HTML, JavaScript and URL features are not specific to mobile and can be used for analyzing desktop webpages as well. However, the mobile features derived from mobile applications such as dialer and SMS do not apply to desktop webpages.

Table 18: Indicators of mobile specific webpages extracted by manual analysis of the top-level mobile and desktop webpages of the 1,000 most popular websites on Alexa. We identified one top-level domain (TLD), nine subdomains and seven URL path prefixes.

Mobile Webpage Indicators	
Top Level Domain	.mobi
Subdomain	m., mobile., touch., 3g., sp., s., mini., mobileweb., t.
URL Path Prefix	/mobile, /mobileweb, /m, /mobi, /?m=1, /mobil, /m_home

5.3.2 Data Collection

Our data gathering process included accumulating labeled benign and malicious mobile specific webpages. First, we describe an experiment that identifies and defines 'mobile specific webpages'. We then conduct the data collection process over three months in 2013.

Identification of mobile specific webpages: We crawled the top-level webpage of the 1,000 most popular websites from Alexa.com [46] using the Android mobile and desktop Internet Explorer (IE) browsers. We used Android mobile version 4.0 and IE desktop version 9.0 for Windows 7. We then manually analyzed each pair of final URLs for the same seed URL when crawled from each browser. Before classifying a URL as mobile specific, we confirmed that the final URLs for desktop and mobile were different for the same seed URL. We also compared the contents of each pair of desktop and mobile webpages, and ensured that the two contents were different. We ignored all the seed URLs that led to an identical final URL when crawled from the desktop and the mobile browser. Our analysis identified nine subdomains (e.g., m.) and seven URL path prefixes (e.g., /mobile) in the URLs of popular websites to represent their mobile specific webpages. Additionally, we considered all URLs with the '.mobi' Top Level Domain (TLD) to be mobile sites [50]. We defined a mobile

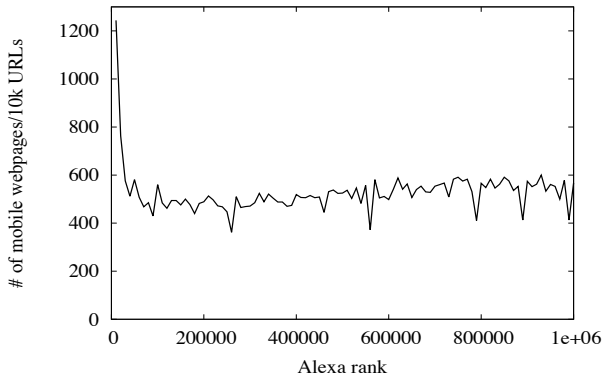

Figure 11: Number of mobile specific websites found in every 10,000 websites in the top 1,000,000 URLs on Alexa.

specific webpage as one containing any of these 17 mobile indicators in the URL and showing differences in content from the corresponding desktop webpage. Table 18 summarizes the mobile indicators.

Building the dataset: To generate training data for our model, we statically crawled the top-level webpage of the top 1,000,000 most popular websites from Alexa from an Android mobile browser. We then extracted the mobile specific webpages using the algorithm described above. Figure 11 shows the number of top-level mobile specific webpages found in the dataset. 1,244 out of the first 10,000 most popular websites offer a mobile specific version and 763 maintain mobile specific webpages in the 10,000-20,000 range. From 20,000 onwards upto one million, the number of mobile specific webpages found using our algorithm is largely constant. We observed that 485 out of the top one million Alexa websites have the '.mobi' TLD. Using the 17 mobile indicators defined in Table 18, we collected 53,638 mobile specific URLs at the top-level by statically crawling each website in Alexa from an Android mobile browser. We then crawled each of the 53,638 mobile specific websites two levels deep. Interestingly, we found links to several non-mobile URLs on the mobile specific

webpages. We discarded all non-mobile webpages and were left with 295,512 mobile specific URLs at depth two. In total, we derived 349,150 mobile specific URLs from the Alexa one million websites.

Gathering data for malicious mobile URLs was challenging since the mobile web is still evolving and new threats are emerging. We monitored several public blacklists [10, 11, 13] continually for a period of three months and extracted mobile specific URLs from the blacklists. We set up a continuous feed from two public blacklists and crawled newly uploaded malicious URLs every two seconds. We also monitored PhishTank's [16] valid and online dataset for mobile specific phishing URLs. After monitoring these sources for three months, we gathered data from 531 top-level and 4,681 depth two mobile specific malicious URLs. Note that our dataset also contains mobile URLs that were submitted to the blacklists before 2013, but were live at the time of crawling.

We established ground truth of the labels (malicious/benign) of webpages in our dataset by using VirusTotal [23] and Google Safe Browsing [43]. The Google Safe Browsing tool performs both static and dynamic analysis on webpages [176]. It first discards benign webpages identified using static analysis and then performs dynamic analysis on the webpages tagged as malicious following static analysis. VirusTotal queries 41 different malware detection tools based on dynamic analysis, crowd sourcing and signatures. To be conservative, we labeled a URL as malicious only when Google Safe Browsing tagged a URL as malicious, or four or more tools queried by VirusTotal labeled the URL as malicious. We also performed manual inspection if necessary. For example, the URLs from PhishTank are crowdsourced, and Google Safe Browsing and VirusTotal do not detect all valid URLs from PhishTank as malicious. We manually visited such URLs to ensure that they are phishing webpages. Our final dataset consisted of 349,137 benign URLs and 5,231 malicious URLs. We used this dataset to train kAYO's model.

5.4 Implementation and Evaluation

We describe the machine learning techniques that we considered to tackle the problem of classifying mobile specific webpages as malicious or benign. We then discuss the strengths and weaknesses of each classification technique, and the process for selecting the best model for kAYO. We build and evaluate our chosen model for accuracy, false positive rate and true positive rate. Finally, we compare kAYO to similar existing techniques and empirically demonstrate the significance of kAYO's features.

5.4.1 Model Selection and Implementation

We treated the problem of detecting malicious webpages as a binary classification problem. We considered each known benign mobile webpage as a negative sample and each known malicious mobile webpage as a positive sample. We considered three popular binary classification techniques in machine learning, Support Vector Machines (SVM), naïve Bayes and logistic regression.

Support Vector Machines (SVM) is a popular binary classifier. However, it works well only on a few thousand samples of data. Due to the scaling problem of SVMs and our large dataset, SVM was not the best choice for kAYO.

Naïve Bayes is generally used when the values of different features are mutually independent. Many features that we extracted were mutually dependent. For example, the number of scripts in a webpage was dependent on the number of internal, external and embedded JavaScript in the webpage, which were three other features of our model. Since the assumptions required for optimal performance of naïve Bayes did not hold for our dataset, we could not use the naïve Bayes classifier.

Logistic Regression is a scalable classification technique and makes no assumption about the distribution of values of the features. Therefore, this technique was the best fit for our dataset. We used the binomial variation of logistic regression to model kAYO and employed $\ell 1$-regularization to avoid overfitting of the data.

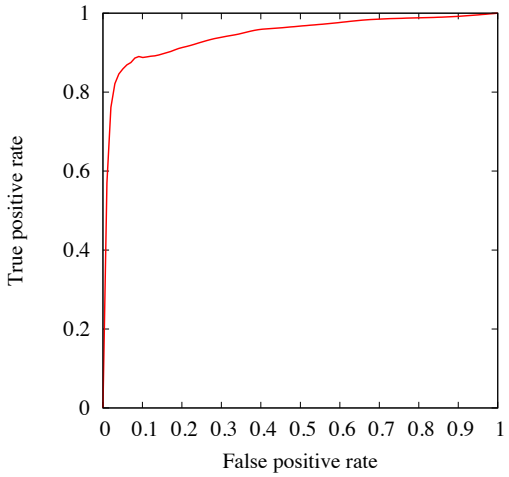

Figure 12: The final ROC curve for kAYO's logistic regression model with regularization.

We used the *scrapy* [19] web scraping framework to crawl the collected mobile URLs. We then built a parser for extracting features discussed in Section 5.3.1 from each input webpage dynamically. The crawler and feature extraction scripts were implemented in Python. We used logistic regression on the extracted features for training and testing. We programmed the logistic regression model in the numerical computing language Octave [8]. We tested the model on a machine with quad core 3.4 GHz Intel Core i7 processor and 16 GB memory.

5.4.2 Evaluation

Our dataset contained 349,137 benign URLs and 5,231 malicious URLs. We divided our dataset into three subsets: training, cross-validation and test. We first randomly shuffled the data and set aside 10% of the data as the test set. The remaining 90% of data was used for training and 10-fold cross-validation.

For each validation round we calculated the accuracy, the false positive rate and the true positive rate on the validation set. We further used $\ell 1$-regularization to avoid

overfitting. We varied the regularization parameter from 0 to 1,000 in the intervals of 10 and chose the best parameter. We then plotted a ROC curve by taking the mean of all false positive rates[6] and false negative rates[7] output from every cross-validation step, and found the best threshold for differentiating between malicious and benign data. Figure 12 shows the final ROC curve.

kAYO provided 91% true positive rate and 7% false positive rate on the cross-validation set. We used the best parameters obtained from the training and cross-validation steps to test the 10% labeled dataset set aside. Our test set shows 90% accuracy, 8% false positive rate and 89% true positive rate. *We anticipate that in reality, the false positive rate on the test set would be lower than what was found using the labeled samples.* This is because kAYO detected a number of malicious mobile URLs in the wild that we hand verified, and were not detected by tools that we used for establishing ground truth of our datatset. More details on examples and in-depth analysis of mobile malicious URLs detected by kAYO in the wild can be found in Section 6.4.3.

Comparison with existing static techniques:[8] We have identified and used 11 new mobile-relevant features previously not studied. We note that none of the existing techniques account for mobile specific features considered in kAYO. The non-commercial static analysis technique closest to kAYO is Cantina [218]. It detects phishing webpages in real-time using static features of webpages. We requested the authors of Cantina to run our test URL set through their tool for direct comparison with kAYO. However, the authors informed us that Cantina was not available. Therefore, we could not compare kAYO's performance directly with it. We instead compare

[6]False positive rate is equal to (1 - recall) or (1 - sensitivity).

[7]False negative rate is also known as precision or specificity.

[8]To the best of our knowledge, kAYO is the first technique that uses static features of webpages to detect malicious *mobile* pages. Therefore, we compare against existing desktop techniques. We also could not secure access to the code or software of these related desktop techniques from the respective authors upon request. Thus, our only option was to base kAYO's comparison on the results discussed in the related research papers of existing techniques.

Table 19: Comparison of kAYO with Cantina, a technique using static webpage features to detect phishing webpages in real-time. kAYO's evaluation set size is over two orders of magnitude larger than that of Cantina. Moreover, kAYO's feature extraction process is two orders of magnitude faster than Cantina. Cantina's functionality is dependent on external tools unlike kAYO and Cantina works well only on webpages written in the English language. kAYO does not have these drawbacks.

Factor	**Cantina** [213, 218]	**kAYO**
Designed to detect	Phishing	Mobile web threats
Detects pages written in	English-only	Any language
Avg. feature extraction time	**2.82 sec**	**0.016 sec**
Evaluation set size (# of webpages)	**200**	**34914**
True positive rate	97%	89%
False positive rate	6%	8%
External dependencies	Requires Google's search engine to function	None
Detects pages missed by Google Safe Browsing?	No	Yes

kAYO with the methodology, speed and performance of Cantina given in related research papers [213, 218]. Table 19 summarizes the comparisons. Cantina provides better true positive rate and comparable false positive rate against kAYO. However, there are several drawbacks to Cantina. First, Cantina's functionality depends on the results of Google's search engine unlike kAYO. Moreover, Cantina assumes that every webpage not ranked by Google is malicious. We argue that this is a strong assumption and might lead to a high false positive rate. Additionally, this methodology prevents Cantina from analyzing webpages not visited by Google's search engine. kAYO does not depend on any external tools and can detect malicious webpages missed by Google Safe Browsing. Second, kAYO's feature extraction process is over two orders of magnitude faster than Cantina. On an average, kAYO takes 0.016 seconds to extract the features of a webpage and Cantina takes 2.82 seconds. We argue that this improvement in the speed of analyzing webpages makes kAYO more usable than Cantina in real-time. Finally, Cantina performs well only on webpages written in

Table 20: Comparison of kAYO with five existing static analysis techniques that detect malicious desktop webpages. kAYO provides the lowest false positive rate on an evaluation set twice as large as the one used by other techniques. kAYO also considers mobile web threats, whereas, the other techniques are focused on detecting desktop web threats.

Technique	**Designed for**		**Tested on**	**False Pos rate**	**Evaluation set size**
	Environment	**Threat**			
[84]	Desktop	Drive by Downloads	Drive by download only	9.9	15000
[188]	Desktop	Malicious JavaScript	Drive by download only	13.7	15000
[153]	Desktop	Spam URLs	Drive by download only	14.8	15000
Union of [153, 188] [108, 147]	Desktop	Drive by, malicious JS, spam URLs	Drive by download only	17.1	15000
kAYO	**Mobile**	**Existing mobile web threats**	**Existing mobile web threats**	**8.1**	**34914**

English due to its heuristic features whereas kAYO can work with webpages written in any language.

We also compared kAYO's performance with existing static analysis tools that detect non-phishing attacks. The closest non-commercial tool to kAYO based on the diversity of features and the scale of the evaluation set is Prophiler [84]. Prophiler detects drive-by-downloads on desktop webpages. We contacted the authors of Prophiler and to allow us to run our URL dataset on their tool. However, they too informed us that their tool was not available. Therefore, we could not perform a direct comparison of kAYO with Prophiler. We instead compare kAYO's performance with the performance numbers of existing static techniques described by Canali et al. [84]. Canali et al. performed an analysis of 15,000 webpages consisting of about 5,000 known webpages launching drive-by-downloads. The contenders of the comparison were then existing tools detecting malicious JavaScript [108, 147, 188], drive-by-downloads [84] and spam URLs [153]. Table 20 and Table 21 show the comparison of performance

Table 21: Comparison of kAYO with five existing static analysis techniques that detect malicious desktop webpages. kAYO's feature extraction process is 10 times faster than the fastest existing technique [188] and classification time is 100 times faster than the fastest existing technique [153]. kAYO is the *only* technique that considers mobile specific features of webpages.

Technique	**Time in sec**		**Considers mobile webpages?**
	Feature extraction	**Classi-fication**	
[84]	3.06	0.24	✗
[188]	0.15	0.034	✗
[153]	3.56	0.020	✗
Union of [108, 147, 153, 188]	N/A	N/A	✗
kAYO	**0.016**	**0.0002**	✓

of kAYO with each of these techniques. kAYO provides the lowest false positive rate over an evaluation set twice as large as the one used by other techniques as shown in Table 20. Moreover, kAYO's feature extraction process is 10 times faster than the fastest existing technique [188] and classification process is 100 times faster than the fastest existing technique [153]. Finally, all the existing techniques are focused on desktop threats, whereas, kAYO focuses on mobile specific threats. Accordingly, had we been able to run these tools over our dataset, they would have performed more poorly.

Need for mobile specific techniques: Because neither Cantina nor Prophiler were made available to us, we performed an experiment to demonstrate the need for new mobile specific models. Intuitively, due to the disparity in the same static features when measured on mobile and desktop webpages (as discussed in Section 5.2), and the emergence of new mobile specific features, a model trained on desktop webpages will not generate precise results for mobile webpages. Note that we are not making claims about the *exact* performance of each system against our dataset; rather, we are attempting to demonstrate (in the absence of either being made available and in good faith) that previously published techniques not considering the changes and new

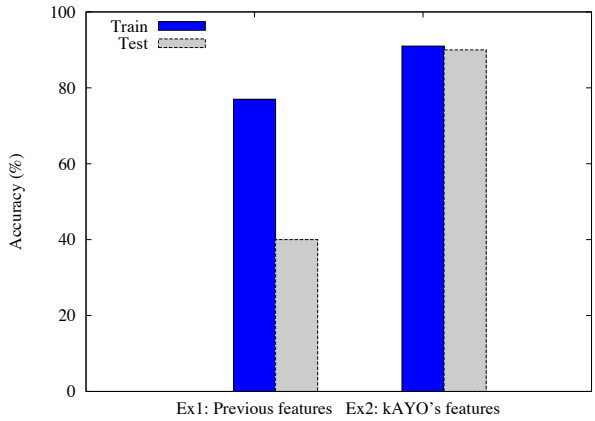

Figure 13: *Ex1*: Results of a model trained on desktop webpages using desktop features studied in earlier techniques and then tested on mobile webpages. *Ex2*: Results of a model trained on mobile webpages by adding mobile specific features to the feature set and tested on mobile webpages. Ex1 shows that a model trained on desktop pages using static features from earlier desktop-specific techniques, when applied to mobile webpages performs poorly. However, when a model is trained with the same static features and additional mobile specific features exclusively on a mobile datatset, the results of testing on a mobile dataset improve significantly as seen in Ex2.

features identified in this work perform significantly worse than our own when analyzing malicious mobile webpages. A more exact comparison would only be available if the authors of these systems make them available.

For this experiment, we created a training dataset of desktop webpages and a test datatset of mobile webpages. We statically crawled Alexa top 10,000 webpages to the second level using the desktop Internet Explorer browser version 9.0 for Windows 7. We obtained the desktop malicious webpages by monitoring public blacklists [10, 11, 13] and crawling live URLs to level two. We verified ground truth of these URLs using Google Safe Browsing and VirusTotal. We randomly shuffled the collected webpages and chose 10,000 webpages while keeping the proportion of benign and malicious webpages in the dataset equivalent to the mobile dataset described in Section 5.3.2. We then created a test dataset of 1000 mobile benign and malicious webpages by

randomly selecting URLs from the larger dataset described in Section 5.3.2.

We extracted 33 out of the 44 static features in kAYO from each webpage in the desktop and mobile datasets. We disregarded the 11 new mobile features used in kAYO and instead focused our analysis on the 33 features previously used in similar desktop static techniques. We note that the goal of this experiment is not to extract all desktop relevant features used earlier, but demonstrate that a model trained on features extracted from desktop webpages does not perform well when applied to mobile webpages. We believe that these 33 features accurately represent the static features used in earlier techniques to detect malicious desktop webpages [84, 108, 114, 147, 153, 188, 213, 218].

We used logistic regression with regularization to train a model on the desktop webpage dataset and tested the model on the mobile dataset. Figure 13 shows the results of our experiments. Ex1 shows that using 33 features, we achieved 77% accuracy in training on desktop webpages. However, when the parameters obtained from this model were applied to the mobile dataset, the accuracy reduced significantly to 40%. *The difference between the accuracy of the training and testing dataset is the important comparison metric in this experiment as it demonstrates the inability of previous desktop-only models to accurately characterize mobile webpages.* Ex2 simply shows kAYO's results (discussed in Section 5.4.2) of training and testing on mobile webpages considering mobile specific features. Both training (91%) and testing (90%) dataset accuracies improve notably. More importantly, the accuracies of the training and testing datasets in Ex2 are comparable unlike those in Ex1. These results further corroborate our intuition that mobile specific static techniques are necessary.

Proprietary techniques: We note that the static analysis component of Google's Safe Browsing tool is proprietary and no information can be retrieved about its performance. Moreover, Google Safe Browsing is not enabled on mobile browsers at present. Therefore, even if Google's tool identifies a mobile webpage as bad, Chrome

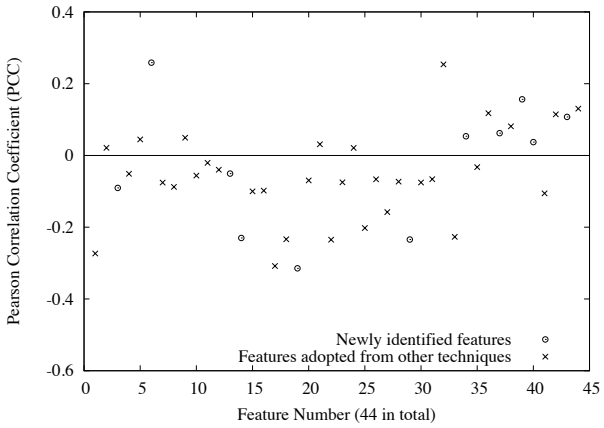

Figure 14: The Pearson Coefficient Correlation (PCC) of each of the features extracted in kAYO with the label (malicious/benign), found using our evaluation dataset. Each point corresponds to the correlation of a feature with the label. In total there are 44 points corresponding to the 44 features of kAYO, including the newly identified features and the ones adopted from existing techniques. The Y value of each point depicts the predictive power of the corresponding feature i.e. PCC. The greater the absolute value of the PCC of a feature, the better predictive power of the feature. Note that all the PCC values are non-zero implying that every feature in kAYO's feature set is significant and impacts the result of classification.

and Firefox mobile browser users do not benefit from this information unlike the desktop users of these browsers. Even though enabling Safe Browsing on mobile is an engineering effort, we later demonstrate the importance of employing a mobile specific technique such as kAYO.

Significance of kAYO's feature set: It is important to observe that kAYO's feature set has been carefully created to ensure relevance to mobile webpages and negligible extraction time. We experimentally demonstrate the significance of kAYO's features using the Pearson product-moment Correlation Coefficient (*PCC*). PCC is a measure of the linear dependence between two variables giving a value between $+1$ and -1 inclusive [180]. In other words, PCC provides information about the predictive power of a feature over the classification result. The larger the absolute value of the PCC of a feature, the more its predictive power. For example, a feature

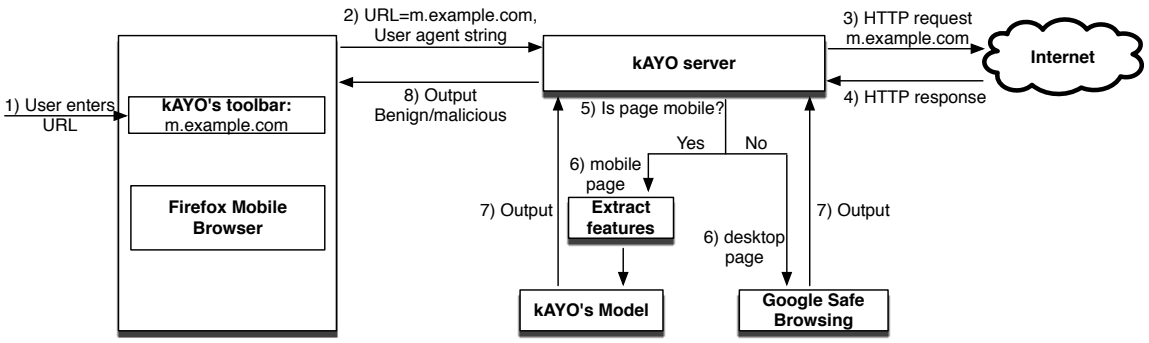

Figure 15: Architecture of the mobile browser extension based on kAYO. User enters the URL he wants to visit in the extension toolbar and receives a response in real-time from our backend server about the maliciousness of the URL. If the URL is benign according to kAYO, the page of interest is rendered in the browser. Otherwise, the user is shown a warning message to not visit the URL.

with PCC -0.6 is a better predictor of whether a webpage is malicious than a feature with PCC 0.21. It is important to note that identifying features with very high PCC values is extremely difficult given the hundreds of different components of webpages and the diversity of possible threats.

We find the PCC between each feature in kAYO's feature set and the label (benign/malicious), from the test set used for evaluation. Intuitively, if kAYO's features are significant, then the absolute value of the PCC of each feature with the label must be non-zero. Figure 14 shows the plot of the PCC of each of the 44 features of kAYO with the label. The circles show the PCC of the newly identified features of kAYO and the Xs depict the PCC of features adopted from earlier works. As seen in the Figure, all the PCC values are non-zero, implying that every feature in kAYO is significant.

Comparison with existing browser tools: Browser extensions and plugins help protect users from visiting malicious websites. The most prevalent threat on the mobile web at present is phishing. Therefore, we surveyed the most popular anti-phishing Firefox desktop extensions for comparison with kAYO. These 33 extensions were selected by searching for the keyword 'phishing' on the Firefox extension store. Most

of the extensions were certificate verifiers, password protectors or file protectors. We did not find any extensions performing content-based static analysis. We disregarded extensions that were built only for one specific website (e.g., FB Phishing Protector and LibertyGuard) or were no longer supported (e.g. Nophish). We then chose the top five extensions (Anti Phishing 1.0, DontPhishMe, Netcraft Toolbar, PhishTank SiteChecker and Phish Tester) based on the number of users for further analysis. We randomly selected a set of 10 known malicious URLs from our dataset and queried each tool with the URLs. PhishTank SiteChecker simply queried PhishTank and returned the result, detecting three of the 10 URLs. Netcraft detected three out of the 10 URLs as well, two of which were also detected by Phish Advisor. Anti Phishing 1.0 detected one URL. Phish Tester and DontPhishMe did not generate any results.

We also tested the freely available trial version of the Lookout safe browsing tool [12]. Lookout is one of the most popular security applications available for mobile devices. This tool protects users of the Android mobile and the Chrome mobile browsers from phishing scams and malicious links on the mobile web. We browsed the same 10 known malicious URLs from both the Android mobile and Chrome mobile browser on a device running the Android 4.0 operating system. *We were presented with alerts for only two out of the 10 URLs by Lookout, while kAYO detected eight out of the 10 webpages.*

Given the paucity of a working extension to detect different threats on mobile webpages, and the unavailability of signature-based tools such as Google Safe Browsing for mobile browsers, we developed a mobile browser extension using kAYO.

5.5 Browser Extension

Building a browser extension based on kAYO adds value for two reasons. First, the mobile specific design of kAYO enables detection of new threats previously unseen by existing services (e.g., pages including spam phone numbers). Second, building an

extension allows immediate use of our technique. We discuss other potential avenues of adopting kAYO in Section 5.6.3.

We developed a browser extension using kAYO for Firefox Mobile[9], which informs users about the maliciousness of the webpages they intend to visit. Our goal was to build an extension that runs in real-time. Therefore, instead of running the feature extraction process in a mobile browser, we outsourced the processing intensive functions to a backend server. Figure 15 shows the architecture of the extension. User enters the URL he wants to visit in the extension toolbar. The extension then opens a socket and sends the URL and user agent information to kAYO's backend server over HTTPS. The server crawls the mobile URL and extracts static features from the webpage. This feature set is input to kAYO's trained model, which classifies the webpage as malicious or benign. The output is then sent back to the user's browser in real-time. If the URL is benign according to kAYO, the extension renders the intended webpage in the browser automatically. Otherwise, a warning message is shown to the user recommending them not to visit the URL.

Users of the extension will browse both mobile specific and desktop webpages since not all websites offer a mobile specific version. Recall that being a mobile specific technique, kAYO does not perform well on desktop webpages. Consequently, processing all pages of interest through kAYO might output incorrect results for desktop webpages. To address this problem, the backend server first detects whether the intended webpage is mobile specific using the same method explained in Section 5.3.2. The webpage is processed by kAYO only if it is mobile. The desktop webpages are analyzed using Google Safe Browsing. Note that any other existing technique for detecting desktop malicious webpages can be used instead of Google Safe Browsing.

We performed manual analysis of 100 randomly selected URLs (90 benign and 10

[9]Firefox Mobile is one of the very few mainstream mobile browsers that support browser extensions. Similar extensions can easily be developed on other mobile browsers once supported.

malicious) from our test dataset and measured the performance of kAYO in real-time. On an average, an output was received in *approximately one second* from the time the user entered a URL in kAYO's toolbar. We argue that the good performance is due to careful selection of quickly extractable features and lower complexity of mobile webpages as compared to desktop webpages. The maximum delay in result generation was seen in scraping the input webpage from its respective server. Caching already scraped webpages with an acceptable expiration time can further reduce this delay. Figure 16 shows a screen shot of our browser extension at work. We plan to make the extension available publicly post publication.

5.6 Discussion

kAYO detected a number of malicious webpages in the wild that were not found by existing techniques. We investigate these webpages in detail and then describe the limitations and future work of kAYO.

5.6.1 Investigating False Positives

We used Google Safe Browsing and VirusTotal for establishing ground truth of our dataset for training and evaluating kAYO. However, such dynamic analysis techniques execute webpages on desktop browsers running on virtual machines and miss mobile specific threats. To validate our intuition, we performed manual analysis on webpages that were identified as malicious by kAYO, but were tagged as benign by Google Safe Browsing and VirusTotal. Performing manual in-depth analysis for all webpages classified as malicious by kAYO was not feasible. Therefore, we chose a random subset of 100 URLs from the false positives obtained by running kAYO on the test dataset in Section 5.4.2.

We verified each of the 100 URLs by visiting them manually from an Android mobile browser version 4.0. We found 10 URLs to be suspicious. These 10 URLs contained survey pages to win iPads or Visa gift cards, uncommon online electronic

equipment stores and stores selling health-related products. Most URLs did not have a Google page rank. One particular webpage prompted a user to download a binary file masquerading as a flash update. We downloaded and found the binary file to be malicious by querying VirusTotal. Another webpage had a known bank fraud phone number prefixed with the `tel:` API. Two out of the 10 suspicious URLs were also marked as suspicious by the Lookout safe browsing tool. We have reported these 10 webpages to PhishTank. Five out of the 10 URLs that we submitted have already been validated by PhishTank and marked as malicious. All 10 URLs went offline within one week of submission. This further strengthens our intuition since phishing URLs are usually short lived [127, 158]. We note that PhishTank might not validate some of the URLs we submitted. This is because, PhishTank's validation process is based on crowdsourcing and threats such as known bank fraud numbers on a website might not be detected without the availability of tools such as Pindrop PRS [17].

5.6.2 Cross-channel threats

We found 173 unique mobile webpages in our dataset (including training and testing) that hosted API prefixed known fraudulent phone numbers and were all tagged as benign by Google Safe Browsing and VirusTotal. These numbers are associated with a number of known financial fraud campaigns (against a number of different major US-based institutions) according to our queries to the Pindrop Security PRS [17]. These results show that adversaries have begun to exploit such cross-channels (e.g., create a phishing webpage and include a fraud phone number) to attack mobile users. Moreover, these experiments suggest that *the false positive rate of kAYO might be lower in reality, given that mechanisms fail to classify such pages as malicious.* We intend to conduct a further analysis of such attacks in our future work.

5.6.3 Limitations and Future Work

The expected concerns of kAYO are similar to those of existing malicious website detection tools using static analysis. Evasion by mimicking the features we consider to be good indicators of a legitimate webpage can be used to defeat kAYO. However, our comprehensive set of features makes it harder to evade kAYO, as seen from our evaluation over a large dataset.

We statically crawled the top million websites of Alexa. Therefore, we did not collect webpages that use JavaScript to detect and redirect to the mobile webpage. We have also missed the mobile webpages represented by ways other than the ones used by the top 1,000 websites. We do not make any claims about gathering all mobile webpages from Alexa top one million. However, given the large set of webpages collected, we believe that our dataset is a representative cross section. Finally, the focus of this work was on mobile webpages designed for phones. We defer the analysis of webpages built for tablets to future work.

kAYO's features reflect current trends in mobile malicious webpages. The potential of bad activity in the mobile web could increase yet further over time. kAYO's feature set and model will need to be updated, according to the new threats faced by the mobile web in the future. However, such updates are necessary in all static techniques that aim to detect new threats.

In-depth dynamic analysis of webpages always provides more accurate results as compared to static analysis techniques. According to our knowledge, there exist no techniques for in-depth analysis of mobile webpages. However, the availability of such tools will enable detection of threats such as malicious mobile binaries hosted on websites. If mobile specific dynamic analysis techniques are developed, then kAYO can be used as a pre-filter to reduce the number of webpages submitted for in-depth analysis. Finally, kAYO can be integrated in existing tools such as Google Safe Browsing.

Using signature based blacklist approaches such as Google Safe Browsing might improve the performance of kAYO's browser extension. A blacklist can be synchronized with kAYO's extension server and enforced locally. Although such techniques might reduce the average delay in page rendering, they will also preclude from protection against webpages that change dynamically defeating kAYO's goal of real-time evaluation. We plan to investigate performance enhancing designs that preserve real-time evaluation in future work.

5.7 Conclusion

Mobile webpages are significantly different than their desktop counterparts in content, functionality and layout. Therefore, existing techniques using static features of desktop webpages to detect malicious behavior do not work well for mobile specific pages. We designed and developed a fast and reliable static analysis technique called kAYO that detects mobile malicious webpages. kAYO makes these detections by measuring 44 mobile relevant features from webpages, out of which 11 are newly identified mobile specific features. kAYO provides 90% accuracy in classification, and detects a number of malicious mobile webpages in the wild that are not detected by existing techniques such as Google Safe Browsing and VirusTotal. Finally, we build a browser extension using kAYO that provides real-time feedback to users. We conclude that kAYO detects new mobile specific threats such as websites hosting known fraud numbers and takes the first step towards identifying new security challenges in the modern mobile web.

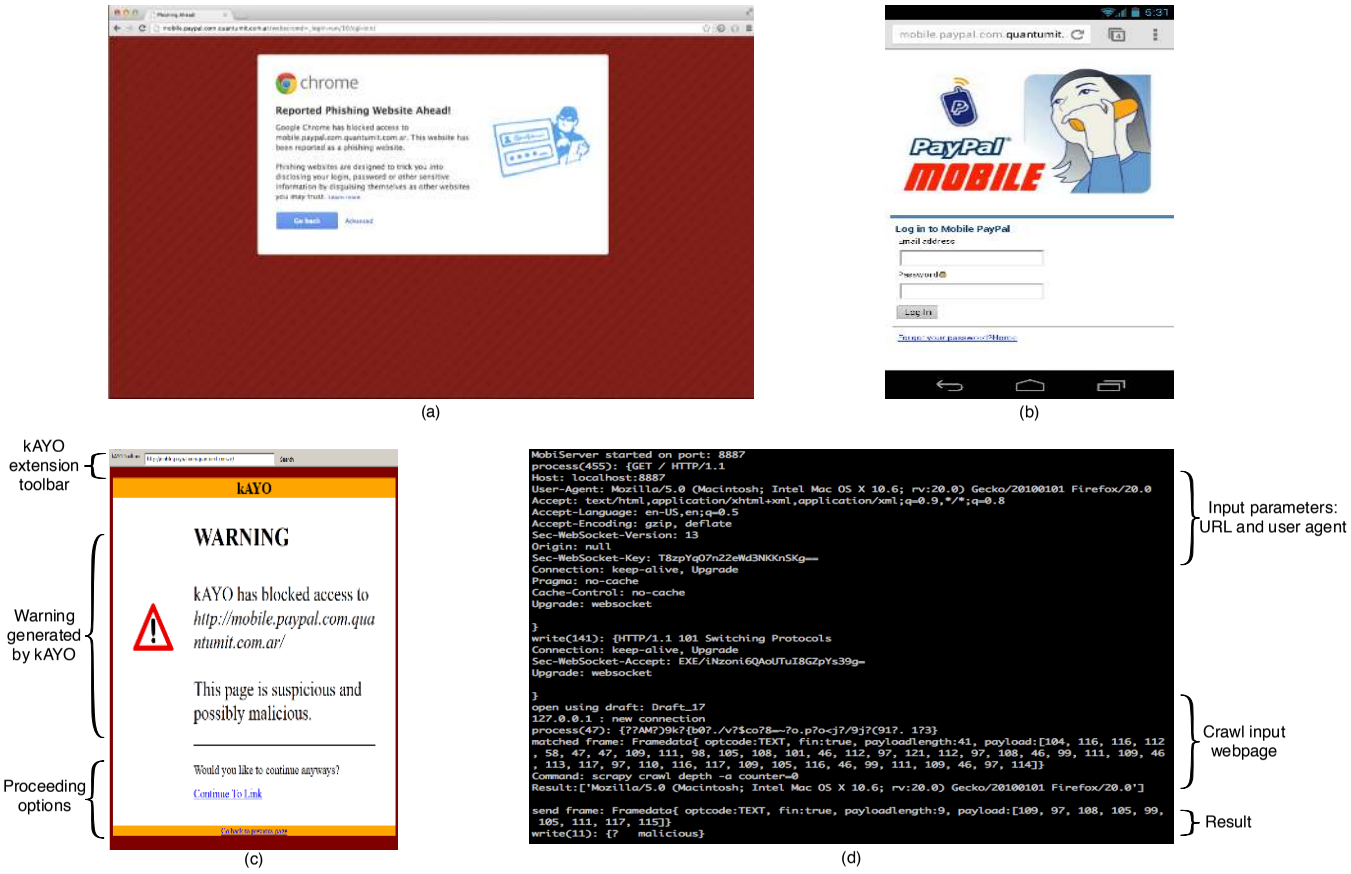

Figure 16: (a): Chrome desktop browser informing the user of a potentially malicious webpage. The webpage is a known mobile phishing webpage. (b): The same webpage when rendered on the Chrome mobile browser, whose users are the real targets, does not provide any warning. (c): kAYO extension running on the Firefox mobile browser detects the webpage as malicious and warns the user. (d): Screenshot of command line processing at the extension server.

CHAPTER VI

FUTURE WORK

This thesis has identified and provided solutions for a number of security issues in modern mobile browsers and webpages. This chapter discusses other open problems in the area of mobile web security. First, we discuss direct extensions of the work presented in this thesis. We then delve deeper into the changing paradigm of mobile applications and propose a permission system for modern mobile web applications.

6.1 Advancing Dialogue on Mobile Browser Security

The work in this thesis demonstrates that mobile browsers have taken steps back in terms of security as compared to desktop browsers. Chapter 4 experimentally illustrated that by and large, desktop browsers adhere to security standards set forth by organizations such as the W3C. However, due to the unavailability of security guidelines specific to mobile browsers, mobile browser vendors have implemented inconsistent and incomplete sets of SSL indicators available on their desktop counterparts. This allows newer versions of mobile browsers to introduce previously non-existent security issues. For example, the content-indicator proximity guideline (3b from Section 4.2.2 in Chapter 4) was correctly followed by Firefox Mobile version 4 Beta 3. However, the latest version of Firefox Mobile (as of June 2013) violates this guideline by placing the padlock icon adjacent to the favicon of a webpage. We strongly believe that such oversights can be avoided by defining and standardizing security guidelines separately for mobile browsers. We plan to advance this dialogue with browser vendors and standardizing institutions such as the W3C to bring strict guidelines for browsers on mobile platforms.

6.2 Tools for Malicious Mobile Webpage Detection

kAYO, the tool presented in Chapter 5 was designed to target existing threats on mobile webpages. The biggest threat on mobile at present is phishing. However, the mobile threat landscape can change in the future. Our tool did not consider threats originating from obfuscated JavaScript since such threats are not yet prevalent in the mobile space. Introducing static features extracted from deobfuscated JavaScript may enable detecting mobile webpages injecting malicious JavaScript. However, introduction of such features will increase extraction time and must be balanced with goals of real-time analysis. Chapter 5 also provided examples of detecting cross-channel threats (e.g., create a phishing webpage and include a fraud phone number) present on the mobile web by querying commercial phone reputation systems. We believe that the mobile specific features related to fraudulent phone numbers are strong. However, the ability of attackers to acquire and advertise new fraudulent phone numbers frequently, opens interesting performance problems for extending kAYO to develop production quality tools. Finally, dynamic analysis techniques will be required to detect webpages hosting malicious mobile binaries or launching drive by download attacks. A combination of static and dynamic tools will present stronger defenses against malicious mobile webpages.

6.3 Hybrid Mobile Applications

The design, structure and languages used to build native apps are usually more complex as compared to web apps. Furthermore, native apps need to be built and modified for each targeted mobile platform. Although mobile web apps provide cross-platform functionality and are less complex than individual native apps, web apps do not yet provide the same device functionality as that of native apps. Therefore, developers often choose the path of hybrid apps. Hybrid apps, similar to native apps, run on the device, and are written with web technologies (HTML5, CSS and JavaScript).

Hybrid apps run inside a native container, and leverage the device's browser engine (not the real browser) to render the HTML and process the JavaScript locally. A web view control is used (e.g. UIWebView on iOS and WebView on Android) to present the HTML and JavaScript files in a full-screen format [99]. A web-to-native abstraction layer allows hybrid apps access to device capabilities (such as the accelerometer, camera and local storage) that are not yet fully accessible in mobile web apps due to the security boundary between the browser and the device APIs.

The dependance of hybrid apps on rendering engines used in existing mobile browsers makes the display security research presented in this thesis applicable to hybrid apps. Furthermore, hybrid apps currently do not provide ways of including user interfaces to show SSL/TLS indicators. This design amplifies the phishing and man-in-the-middle security problems discussed in Chapter 4.

Other future research problems include building privacy preserving contextual permission systems for hybrid applications. Existing efforts in this area [190] take the first step by building a manifest-based permission system that controls application behavior using information flow control. Developer learning curve and user incentive of understanding and defining a complex set of rich policies are some of the limitations of existing works. We plan to delve deeper into this area. In particular, we plan to investigate challenges in building unified permission systems that can be deployed for web, native and hybrid apps on mobile platforms.

6.4 Unified Permission Systems for Mobile Web Apps

The family of HTML5 technologies is set to dramatically change the way in which applications are designed for mobile devices. In particular, HTML5 provides direct support for features including audio, video and geolocation information. While access to these features has become a mainstay of "native" mobile apps, their inclusion in this standard makes it possible for mobile web apps to provide many of the features

currently implemented by their native counterparts. Direct support for these features within mobile browsers will be transformative - whereas app makers have traditionally had to invest significant effort to develop software across multiple platforms, HTML5 will allow developers to rely on the mobile browser to deliver a single codebase and a unified user experience.

Mobile web browsers provide web apps with access to features including cookies, Javascript, native code and Flash by default. This may lead to granting more than necessary privileges to certain web apps. For example, a file-sharing app has access to Flash by default but may not need it for proper functioning. Access to the default browser features, potentially sensitive hardware (e.g., camera, microphone) and data (e.g., GPS location, contact information) require protection when provided to web apps. While support for HTML5 features is currently limited in mobile web apps, desktop browsers providing such features typically prompt users on a per-use, per-site basis. Mobile browsers too have adopted the same per-use, per-site permission model and thus, suffer from several weaknesses. The current permission model for mobile web apps does not provide a holistic view into the permissions required by an app. Providing a single interface containing every permission that may be used by an app allows both users and security experts a better opportunity to assess the potential for malicious behavior by the app. As an example, the Android manifest file and install-time warnings have successfully served as the basis of a wide-range of malware-detection tools [86, 89, 107, 109, 111, 219, 220]. However, we note that simply extracting the underlying platform's (such as Android) model and mapping web apps' requests directly to the platform APIs [189] is not the best matching since the structure of mobile web apps and native apps is different. We discuss several differences in the factors contributing to the security of native apps and web apps. We then argue that there is a need to define permissions for web apps separately while maintaining as much overlap with native app permissions as possible for user

learnability and reducing developer efforts. Finally, we argue that the dynamic nature of mobile web apps necessitates a one-stop, easy-to-use permission model that can allow users to access, modify or change the permissions granted to individual web apps.

In this work, we present a proof of concept mechanism that addresses some of these weaknesses and allows a subset of the Android application security model to be easily expressed by remotely stored mobile web apps. Our goal is to provide expert users with a single interface that allows them to reason about the permissions requested by mobile web apps similar to native apps. We start with a clean slate with no permissions given to a web app by default. We argue that developers should be required to declare permissions needed for a web app up front including the permissions provided by default in current browsers. We then discuss a hybrid approach of install-time and run-time permission authorization. We propose *webifest*, an XML file similar to the manifest in Android framework that includes permission declarations and can be used to allow mobile web apps to provide a concise declaration of the resources they intend to use. We argue that encouraging mobile web app developers to request fewer privileges will reduce the attack surface.

We are careful to note that our proposed solution is *not* a tacit endorsement of the Android permission model in particular. Specifically, we are not arguing that users fully pay attention to and understand all Android permissions. Should a better model be found, *we would still argue that all mobile applications should be evaluatable through a security interface providing a complete view of potential behavior.* Given its extensibility and the success with which security experts have had using it to detect malicious applications, we simply rely on the Android model to illustrate our point.

The remainder of this section provides a brief overview of our proposed architecture and attempts to consider best practices for webifests.

6.4.1 Background

We discuss factors affecting security of mobile web apps and compare the significance of the same factors in the security of native apps. Building on this analysis, we discuss the potential requirements in defining a permission model for mobile web apps.

6.4.1.1 Security Factors

Web apps provide cross-platform functionality, reduce developer effort and are easy to update. However, these useful properties also make web apps difficult to secure. We discuss several factors that contribute to the security of a web app and compare the security consequences of each of the factors with those of native apps.

- Nature of permissions: Mobile web apps are more dynamic in nature because web application providers can easily update the server side code. This dynamic nature of web apps allows frequent changes to the permissions required for execution. In comparison, it is difficult to silently make such changes to native apps. Current web browsers also do not provide an interface to view all the permissions required by a web app, unlike a native app.

- User effort: One of the primary protections for native apps is the attacker has to lure the user into installing his app. It is much easier for an attacker to lure a user into following a link to the attacker's web app through email or advertisements.

- Application markets: Native apps have some level of security through app markets (such as Apple and Google) that detect and remove malware. No such mechanism exists for mobile web apps. Unless an online marketplace such as the Chrome webstore [21] is formed for the mobile web, having a centralized system for security analysis will not be possible. Moreover, if such a system is developed, the dynamic nature of web apps will require continuous monitoring

to ensure security. Therefore, permission/capability detection during application submission to the marketplace (e.g., the MSIL code analysis performed on the Windows Phone Marketplace [52]) may not work well for mobile web apps.

- Identical app logic: It is not possible to ensure that all users of the same web app receive an identical copy of the app unlike native apps. Therefore, web apps can collect contextual data such as location more easily by asking different permissions from different users. Due to all users possesing an identical copy of a native app, user-rating systems in native application markets work well. These ratings allow an average user to decide whether an app is good or bad. No such protection exists for mobile web apps. Moreover, even if a third-party entity such as Google independently discovers that a particular mobile web app is bad, conveying this information to the future visitors of the web app is difficult.

- Default permissions: Android native apps are provided with no default permissions. Web apps running in a browser are provided with permissions to access several browser resources such as cookies (maintaining SOP), Flash, download code to the device and run Javascript. Therefore, certain web apps may end up with more privileges than required for their execution.

This is not a comprehensive list of all the factors associated with the security of web and native apps. Other factors including security of the underlying operating system [121], browser [57, 61], identity management using certificates also impact mobile web app security. We have discussed the major factors that lead to differences in the security of web and native apps.

6.4.1.2 Reflection

The comparison between native and web apps shows that additional security factors need to be considered while designing permission systems for managing access requests from mobile web apps. However, we also observe that following wide adoption

of HTML5, both native and web apps on mobile platforms will request access to similar user sensitive data and hardware. Therefore, we choose an approach that will build upon the permission model for native apps and also provide additional features required by mobile web apps.

We anticipate more requests to sensitive information from a mobile web app as compared to a desktop web app. This is because a mobile device can provide contextual information such as location unlike desktop. Current desktop and mobile browsers request run-time permissions for each feature requested by a web app. Desktop browsers such as Safari allow permissions to be stored for a specific time duration. Other browsers such as Chrome desktop store the location permission given to a website forever unless the user revokes it. However, current browsers do not allow a user to view all the permissions used by a web app nor do they allow a user to easily access, modify or revoke permissions.

The lack of a centralized authority such as an app market increases the probability of malicious web apps on the mobile platform. We argue that mobile web app users should be able to easily revoke or selectively grant permissions without being overwhelmed with warnings and thus suffering from warning fatigue.

6.4.2 Proposed Architecture

Goal: We want to provide expert users with a single interface that allows them to reason about the permissions requested by mobile web apps similar to native apps.

We use the Android system as an example and argue that our central idea can be applied to any mobile platform. Our proposed model [60] provides a user consent permission system that gives full view of the required permissions, alerts the user when a web app asks for dangerous permissions and also allows easy revocation of permissions granted to individual web apps. We propose that a web app developer declares all the required permissions to the browser using an XML file similar to the Manifest

file in Android. In addition to specifying *how* to access a particular phone feature, our model requires a web developer to define *what* he wishes to access in the form of permission. For example, when a web developer uses the HTML5 Geolocation API to access location, he should specify that the app would require the corresponding permission ACCESS_COARSE_LOCATION as defined in the Android framework.

Permission categorization: We propose a design where the set of permissions given to a web application by default is null. Therefore, to access resources such as Flash, a web developer will have to request permission. This behavior is significantly different than the behavior of current browsers where a website rendered in the browser runs with full browser-privileges and only requests user permission for access to features such as location. We note that we always allow web apps access to the core platform technologies defined in the browser technologies for HTML5[1] [128] and do not consider them in the permission system. The motivation behind requiring developers to explicitly request access to default browser features such as Flash is to encourage web developers to request least privilege. However, the current per-use, per-site permission model would generate multiple warnings every time a user accesses a web app if universally implemented features such as cookies have to be authorized. This may lead to warning fatigue [35, 103, 187, 195] and careless clickthrough.

We propose classifying permissions required by mobile web apps into two categories, normal and dangerous. We argue that a web app connected to the Internet should not be allowed access to permissions corresponding to the `signatureOrSystem` category [5] in the Android permission system. We also note that permissions in the `signature` category [5] loosely correspond to the already implemented Same Origin Policy in browsers. Out of the 75 normal or dangerous [145] permissions provided by the Android framework, warnings are generated only for the dangerous permissions at

[1]The HTML5 core platform technologies are HTML, CSS, DOM and Javascript.

install-time. We argue that permissions required by web apps should also be treated in a similar fashion. However, *we note that the set of permissions for web apps falling under the normal and dangerous categories would differ from the ones defined in the Android framework and not all of the permissions for native apps on Android would be relevant to web apps.* For example, permissions such as BROADCAST_SMS and BROADCAST_PACKAGE_REMOVED are not relevant for web apps. Additionally, permissions such as INTERNET are crucial for a web app to work unless it is working in offline mode. Therefore, we envision that the number of permissions that a web app can request would be much lower than 75. Even if the browser produces run-time warnings for time-of-use dangerous permissions, we envision that the number of warnings would be limited.

We propose categorizing permissions based on whether a permission will provide access to a user's private data. Normal permissions would be the ones that are crucial for the basic functionality of a web app. Examples of a normal permission are INTERNET, ACCESS_WEBSTORAGE etc. Examples of dangerous permissions would be CAMERA, CALL_PHONE and RECORD_AUDIO. The normal permissions will be granted by the browser without user consent while a warning will be generated when a web app requests access to a dangerous permission.

Managing webifests: We propose a *webifest*, an XML file that can be used by web developers to define all the permissions required by an app. The browser intercepts all webifest files sent by the website in the top-level address bar. For example, consider the following webifest:

```
<webifest domain=foo.com>
<permission=INTERNET, ACCESS_COOKIES,
CAMERA, ACCESS_COARSE_LOCATION>
</webifest>
```

This webifest indicates that a web app from the domain `.foo.com/` is requesting permissions to access the Internet, cookies, camera and coarse location.

When a web app is loaded, the browser intercepts the webifest file only if it is sent over an HTTPS connection. This is to avoid a man-in-the-middle attacker changing the permissions requested by a website. The browser then searches existing webifests for one that matches the domain of the new webifest. If an old webifest does not exist, the browser parses the permissions in the webifest into the normal and dangerous categories. The browser provides normal permissions to the web app without user consent.

Instead of generating an install-time warning for all the dangerous permissions at once similar to native apps, the browser then uses the run-time warning model. When a web app requires a dangerous permission, the browser generates a warning and the user has to approve the permission. Note that the browser does not store the approval when user consents through a warning message. However, the browser provides an interface for the user to selectively approve, store or revoke permissions by using an interface users are already familiar with. We propose extending the interface provided by browsers to access SSL certificate information to accommodate permissions. For example, consider the Chrome mobile browser shown in Figure 17. Clicking on the lock icon in Chrome mobile opens a dialogue that provides identity information of a website. A Chrome mobile browser running our proposed model will also provide an interface to store or revoke *any* of the permissions requested by a web app. When the browser intercepts a webifest, it generates a list of permissions requested by the web app. This list contains both the normal and dangerous permissions and the status information of whether they have been approved. Once a user 'stores' a permission, the browser retains the consent and the user is not asked to approve permissions on subsequent visits to the same web app until the webifest is either revoked by the user or modified by the corresponding web app.

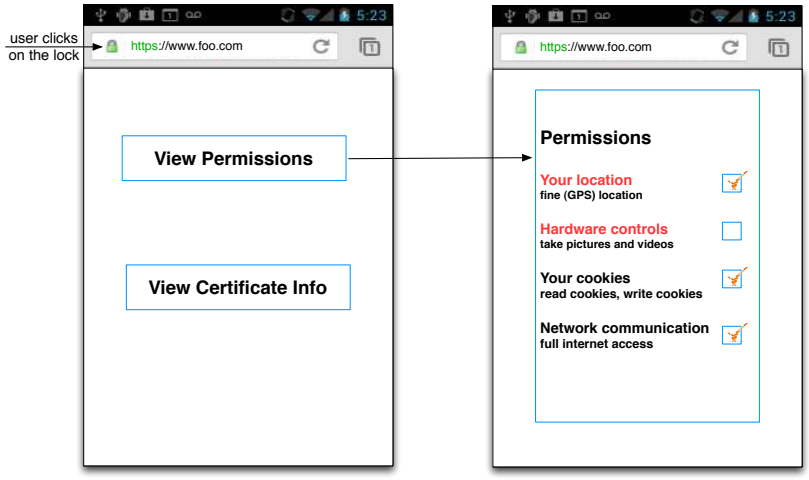

Figure 17: User interface for permission management of mobile web apps. **Left:** When a user clicks on the lock icon, the browser shows this interface to interact with permissions and certificates. **Right:** The browser provides an interface to view the status of the permissions requested by `www.foo.com` (domain in the address bar). The user has not stored hardware controls permissions, whereas he has stored the location, cookies and Internet access permissions. The cookie and Internet permissions are normal permissions granted without user consent. The location and hardware control permissions require explicit user consent. The user can easily revoke a permission by unchecking the box next to it.

If a browser finds an older webifest for the same 'domain' as that of a newly received webifest, the browser compares the values of the 'Permissions and Domain' attributes of the old and the new webifest. If the attributes are the same, the browser simply ignores the new webifest. Otherwise, the browser generates a warning for the user about a new webifest sent by the web app and displays the additional permissions required. If the user authorizes the new webifest, the browser stores the new one and discards the old copy. The browser still requires run-time approval of additional permissions if any in the new webifest. If a user rejects the new webifest, the web app continues to execute with the permissions in the old webifest until the user deletes the webifest.

We note that our proposal is only for a website requesting permissions of the underlying system. However, we observe that in addition to requesting necessary

permissions, a website can also register web-intents with the browser using the webifest file, similar to the manifest file in native apps.

Storing webifests: We propose that a webifest for each web app should be stored using the HTML5 web storage [207]. Web storage supports 'local storage' which is similar to persistent cookies. When a browser intercepts a webifest, the webifest file is stored in the local storage allocated to the corresponding web app's domain. The browser also maintains the status of user approval on all the permissions. Javascript is not allowed access to a webifest unlike other local storage objects.

Managing access requests: When a web app requests access to a phone feature such as camera, the browser searches for the corresponding webifest. The browser verifies whether the user has already granted the requested permission and if granted, allows access without user intervention. If a webifest corresponding to a web app does not exist or the user did not store consent to access the resource requested by the web app, the browser does not allow access. If the permissions required by a web app change, the web app is expected to send a new webifest with the modified permissions in the HTTP response header.

Revocation: Revoking permissions from individual web apps is straightforward. To revoke all the permissions, a user can simply delete the corresponding webifest from the local storage of an app. Alternatively, the user's browser is required to provide an interface that allows the user to revoke all permissions using the in-browser interface. A user can also selectively revoke permissions granted to individual web apps using the in-browser interface shown in Figure 17. Current mobile browsers do not provide an interface to clear access granted to individual websites. For example, in the Android, Dolphin, Firefox Mobile and Opera Mini browsers, a user is required to

revoke location access from all web apps at once. This browser behavior precludes a user from revoking permissions granted to only one app if he has provided location authorization to multiple web apps such as Yelp and Google maps. Existing browsers will need modifications to support the proposed revocation procedure.

Conditions: We require that a browser supporting our *initial* model allow only the domain displayed in the address bar of a web app (top-level domain) to save a webifest. Secondly, a cross-domain element embedded in the web app is prohibited from requesting access to hardware or data.

The reasons for allowing only the top-level domain of a web app to save a webifest are the following: a web app may contain cross-domain embedded elements such as advertisements in iframes. If a browser processes the webifest received in the HTTP response header of such an element, there are two ways of obtaining user authorization for the permissions requested by the embedded element. The browser can create separate authorization warnings for the webifest received from each domain or the browser can combine the permissions requested by all domains and create one warning listing all the permissions and the respective domains. The former can overwhelm the user and latter may provide a false sense of security. In the latter scenario, a user may perceive the set of permissions listed in the long warning as the permissions required by the top-level web app to execute. Prohibiting embedded elements from saving webifests can address these issues. Therefore, a browser supporting our model ignores webifests received from embedded elements in a web app.

The second condition of disallowing a cross-domain embedded element to request access to hardware or data follows as a consequence of enforcing the first condition. Since our model requires a webifest for every 'domain' to be able to access a resource, if the browser ignores webifests received from embedded elements, the corresponding domains are unable to access resources.

Consider a web app at `foo.com` that saves a webifest in the browser when the user visits the web app. Later, if an iframe from `foo.com` is embedded in another web app accessed by the user, `foo.com` may try to access resources using the already existing webifest. To avoid this, we require that the browser ignore access requests made by cross-domain elements embedded in the top-level web app. Cross-domain embedded elements only have access to the Internet. We explore the consequences of these conditions in the discussion section.

Size and content: The expected memory overhead in integrating the proposed model in current browsers would be insubstantial due to the small size of webifest files.

Current web apps in mobile browsers do not require all the information provided in the manifest file for native apps for their execution. In addition to declaring required permissions, the manifest file in a native app describes application components such as activities and broadcast receivers, declares permissions required by other apps in order to access the app, the libraries that the app is linked against and the minimum level of the Android API required by the app [20]. Most of this additional information is required since a native app code resides on the device and is always available. This is not true for web apps. For example, at present there is no mechanism that enables a web app to handle an intent created by a native app or the Android system. Whether such a mechanism is possible is an interesting project beyond the scope of this work. Since the compelling need for current mobile web apps is effectively handling permissions, we chose a lightweight manifest file.

6.4.3 Discussion

The goal of the proposed model is not necessarily to ensure that average users make better security decisions, but instead to provide an interface for expert users to be

able to assess the potential behavior of web apps. If a security expert is unable to evaluate whether a web app is malicious, how can an average user be expected to do the same? Nevertheless, we believe that maintaining significant overlap between the permission models for native and web apps will help average users in making informed security decisions. More importantly, security experts will be able to use the proposed model to design tools similar to the malware detection tools for native apps [86, 89, 107, 109, 111, 219, 220].

There are other efforts in the area of permission management for mobile web apps such as Mozilla's WebAPI [44]. However,WebAPI aims at providing consistent APIs that will work in all web browsers and our motivation is designing a permission model for mobile web apps that increases security and user control. Another related effort in restricting the capabilities of web apps is the Content Security Policy (CSP) [194]. CSP enables the authors of a web app restrict from where the application can load resources, whereas our proposal deals with restricting web apps' access to the resources on a user's device. We discuss the advantages and disadvantages of the proposed model.

Pros: Although the normal permissions are granted without user intervention, their use will enable developers to request least privilege. For example, the default Android mobile and Opera Mini browsers support Flash and videos can be played easily in the browser without user consent. However, a word to PDF converter web app may not require Flash support. Requiring a web developer to ask for Flash permission explicitly may reduce overprivileged web apps.

The simple update and revocation procedure for permissions granted to web apps maintains the highly dynamic nature of web apps while allowing the user to revoke permissions easily at will. For example, if a security expert wants to disable cookies for a suspicious looking webpage, he can do so using the selective permission revocation

model (desktop browsers have a similar functionality for cookies). The user will not have to disable cookies across all webapps in this case, a provision currently available in mobile web browsers. An average user on the other hand will not have to worry about basic details of browser management due to the normal permissions being provided by the browser without user consent.

Providing a similar user interface for permission management as that of the native apps facilitates user learnability of the proposed web app permission model. Moreover, striving for maximum overlap between the permissions defined for native and web apps may reduce the effort required to secure mobile apps (native and web) and also reduce developer effort.

Cons: Prohibiting a cross-domain embedded element such as iframe from storing webifests may break the logic of certain mobile web apps. However, we argue that due to the constrained nature of mobile browsers, the complexity of mobile web apps is lower. Therefore, the number of cross-domain embedded elements in mobile web apps is expected to be minimal. We plan to investigate models allowing a cross-domain embedded element to access resources. One such method would be to fall back to the per-use, per-website model currently used in browsers. For example, if a non-Google web app includes a Google map, the browser can generate an authorization request for location access when a user wants to interact with the map and never provide the interface to store the permission. Another potential technique would be to fetch a webifest file when a user interacts with a cross-domain embedded element. For example, if a non-Google web app includes a Google map, when a user wishes to interact with the map by clicking on it, the browser fetches the webifest file for the map and generates an authorization request to access the resources required by Google maps, again with no storing facility.

Our model mandates explicit authorization for cookies and ignores webifests sent

by embedded elements. This prevents third-party cookies in a webpage from tracking a user across sessions. However, due to the universal usage of cookies, not allowing cookies from other domain excluding the top-level domain in the address bar may break several websites. Whether our model should allow cookies for embedded elements is a topic for future work.

If a web application does not support webifests, a browser supporting webifests can default to the current model of processing permissions based on per-use, per-website. Although this approach maintains backward compatibility, it defeats the purpose of our model. We assert that a browser should support the same permission model for all the elements of a web app irrespective of their domain. If the top-level web app supports webifests, the browser should use the webifest model for all the elements. Otherwise, the browser should use the per-use, per-website model.

Adopting the proposed model will require collaborative efforts from web developers and browser vendors. Finally, storing the webifests file may be tricky since implementing local storage features in a browser may pose information leakage or information spoofing risks [207].

Security comparison with native apps: The notion of security in the proposed model is similar to that of the permissions model for native apps. The model does not protect users against malicious apps that can access sensitive data as a result of user authorization. The level of security depends on a user's knowledge about permissions and the consequences of authorizing a web app to access sensitive information. Moreover, the model cannot provide the guarantee that a web application would request the least privileges required for its execution. Researchers have previously shown that native app developers attempt to obtain least privilege for their applications, but fall short due to API documentation errors and lack of developer understanding [109]. We expect the same effort from mobile web app developers. We also note that developers

that change the permissions required by their app too often might turn away users. Since the browser does not store a permission unless explicitly approved by the user and also alerts the user when a new webifest is detected, multiple changes in webifests may make the user suspicious or annoyed.

Research questions: Several research questions present themselves:

- Normal versus dangerous permissions, where to draw the line? We based our initial proposal of categorization on whether a web app requests permissions to access a user's private data. Are there other possible categorization procedures that can further reduce warnings?
- If the set of normal permissions is large, would it be problematic to grant normal permissions without user consent?
- Would users and developers understand the new model easily?
- Can a mechanism other than local storage be used to maintain webifests? Using local storage requires creating a special exception for preventing Javascript from accessing webifests.
- Is warning fatigue possible for the proposed model? If the total number of permissions available for web apps on a platform is limited, the probability of warning fatigue would be curtailed.

Looking forward: Due to the availability of contextual information, we envision widespread use of HTML5 to access user's data on a mobile device. We have taken the first step towards rethinking the permission system for mobile web apps. An alternative permission model would be adopting the install-time all-or-nothing model defined in Android native apps. Yet another client-side solution for the permission management problem would be defining in-browser policies to limit a user's access to

potentially malicious websites and also block a website's access to sensitive features such as contact lists. Although this approach is possible for corporate phones, it would be difficult to implement on personal phones of average users. Moreover, restricting functionality can result in poor user experience. To avoid pushing security decisions to the user, multiple security tools such as Zozzle [94] can be executed in the browser to protect the user from malicious web apps. However, due to the hardware limitations of mobile devices, this approach will entail severe performance penalties and will not be practical. Finally, if a mobile web app store similar to the Chrome webstore is available in the future, providing centralized security measures would be possible.

Our proposed model provides more control to the user, but also redirects more security decisions to the user. Offloading security decisions to a user is not the best idea. However, we imagine that in the absence of a proxy like setting or an umbrella webstore, client side defenses on mobile devices would be limited.

CHAPTER VII

CONCLUSION

Mobile web browsers and webpages are significantly different than their desktop counterparts. This dissertation demonstrated that several factors impact the security of mobile web browsing including porting of desktop browser code to the mobile environment, usability changes made to browsers to adapt to the small screen, modifications in the content and layout of mobile specific webpages, and rich functionality offered by mobile webpages (or web apps). We then discussed several new pain-points in securing mobile web browsing by identifying new vulnerabilities in modern mobile browsers, experimentally demonstrating the differences between mobile and desktop browsers in terms of security, illustrating the tension between usability and security on the small screen platform, and showing the need for new mobile specific tools to detect malicious webpages. Finally, this work provided potential solutions for the new found vulnerabilities, successfully initiated a dialogue with browser vendors to address these security issues, and built the first mobile-specific static tool to detect malicious webpages in real-time.

This dissertation is just the first step towards securing mobile web browsing, an activity performed by hundreds of millions of people everyday. Looking forward, inclusion of new features in mobile webpages, adoption of HTML5 and WebAPI suites such as Boot2Gecko, existence of native, hybrid and web apps on the same mobile platform, and further increase in the use of the mobile web for sensitive communications will lead to new challenges in securing mobile web browsing.

REFERENCES

[1] "150 Highest Paying Adsense Keywords Revealed!." http://earns-adsense.blogspot.com/2008/04/150-highest-paying-adsense-keywords.html.

[2] "Android Open Source Project - Issue Tracker. Bug #56715." https://code.google.com/p/android/issues/list.

[3] "Android Open Source Project - Issue Tracker. Bug #56716." https://code.google.com/p/android/issues/list.

[4] "Android Open Source Project - Issue Tracker. Bug #56720." https://code.google.com/p/android/issues/list.

[5] "android permission categories." http://developer.android.com/guide/topics/manifest/permission-element.html.

[6] "Chrome, Firefox get clickjacked." http://www.zdnet.com.au/chrome-firefox-get-clickjacked-3392 94633.htm/.

[7] "Facebook clickjacking." http://personalmoneystore.com/moneyblog/2010/08/18/facebook-clickjacking-social-network -scams/.

[8] "Gnu octave: high-level interpreted language." http://www.gnu.org/software/octave/.

[9] "GoDaddy SSL certificate." http://www.godaddy.com/Compare/gdcompare_ssl.aspx?isc=sslqgo016b.

[10] "hphosts, a community managed hosts file." http://hphosts.gt500.org/hosts.txt.

[11] "Joewein.de LLC blacklist." http://www.joewein.net/dl/bl/dom-bl-base.txt.

[12] "Lookout." https://play.google.com/store/apps/details?hl=en&id=com.

lookout.

[13] "Malware Domains List." http://mirror1.malwaredomains.com/files/domains.txt.

[14] "Mobile Browser Market Share." http://gs.statcounter.com/#mobile_browser-ww-daily-20120307-20120405.

[15] "Paying by the Click." http://www.nytimes.com/2007/10/15/us/15bar.html?ref=us.

[16] "Phishtank." http://www.phishtank.com/.

[17] "Pindrop phone reputation service." http://pindropsecurity.com/phone-fraud-solutions/phone_reputation_service_prs/.

[18] "Same-origin policy." http://code.google.com/p/browsersec/wiki/Part2#Same-origin_policy.

[19] "Scrapy | an open source web scraping framework for python." http://scrapy.org/.

[20] "The AndroidManifest.xml File." http://developer.android.com/guide/topics/manifest/manifest-intro.html.

[21] "The Chrome Webstore." https://chrome.google.com/webstore/category/home.

[22] "VeriSign certificate." https://www.verisign.com/ssl/buy-ssl-certificates/index.html?sl=t72010166130000002\&gclid=CIKMyY2GuKgCFYg32godV2_8Bw.

[23] "VirusTotal." https://www.virustotal.com/en/.

[24] "Web-based Android attack." http://www.infoworld.com/d/security-central/security-researcher-releases-web-based-android-attack-317?source=rss_security_central/.

[25] "Key words for use in RFCs to Indicate Requirement Levels." http://www.ietf.org/rfc/rfc2119.txt, March 1997.

[26] "The Transport Layer Security (TLS) Protocol Version 1.1." http://www.ietf.org/rfc/rfc4346.txt, April 2006.

[27] "Dereferencing HTTP URIs." http://www.w3.org/2001/tag/doc/httpRange-14/2007-05-31/HttpRange-14, May 31 2007.

[28] "Overflow clickjacking." http://research.zscaler.com/2008/11/clickjacking-iphone-style.html, November 2008.

[29] "Guidelines for the Processing of EV Certificates, version 1.0." http://www.cabforum.org/Guidelines_for_the_processing_of_EV_certificatesv1_0.pdf, January 2009.

[30] "SSLstrip, presented at Black Hat DC." http://www.thoughtcrime.org/software/sslstrip/, 2009.

[31] "Android Browser Exploit." http://threatpost.com/en_us/blogs/researcher-publishes-android-browser-exploit -110810, 2010.

[32] "Clam AntiVirus." http://www.clamav.net/, 2010.

[33] "Gartner outlines 10 mobile technologies to watch in 2010 and 2011." http://www.gartner.com/newsroom/id/1328113, 2010.

[34] "Guidelines For The Issuance And Management Of Extended Validation Certificates, version 1.3." http://www.cabforum.org/Guidelines_v1_3.pdf, November 20 2010.

[35] "W3C: Web Security Context: User Interface Guidelines." http://www.w3.org/TR/wsc-ui/, August 2010.

[36] "Baseline Requirements for the Issuance and Management of Publicly-Trusted Certificates, version 1.0." http://www.cabforum.org/Announcement-Baseline_Requirements.pdf, April 11 2011.

[37] "Comodo compromise." http://www.csoonline.com/article/678777/comodo-compromise-expands-hacker-talks, April 1 2011.

[38] "DigiNotar CA compromise." http://community.websense.com/blogs/

securitylabs/archive/2011/08/30/diginotar-ca-compromise.aspx, August 30 2011.

[39] "Mobile Browser Market Share." http://gs.statcounter.com/#mobile_browser-ww-monthly-201011-201111, November 2011.

[40] "Opera Presto 2.1 - Web standards supported by Opera's core." http://dev.opera.com/articles/view/presto-2-1-web-standards-supported-by/, 2011.

[41] "The CA/Browser forum." http://www.cabforum.org/, April 11 2011.

[42] "The WebKit Open Source Project." http://webkit.org/, 2011.

[43] "Google developers: Safe Browsing API." https://developers.google.com/safe-browsing/, 2012.

[44] "Mozilla WebAPI." https://wiki.mozilla.org/WebAPI, July 2012.

[45] "Study: Most mobile web browsers unsafe." http://www.upi.com/Science_News/Technology/2012/12/05/Study-Most-mobile-Web-browsers-unsafe/UPI-73431354743353/#ixzz2EGtQsuLd, 2012.

[46] "Alexa, the web information company." http://www.alexa.com/topsites, 2013.

[47] "Apache Tomcat 7.x vulnerabilities." http://tomcat.apache.org/security-7.html, 2013.

[48] "Apps created with phonegap." http://phonegap.com/app/, 2013.

[49] "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2012–2017." http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.html, 2013.

[50] "dotmobi. internet made mobile. anywhere, any device.." http://dotmobi.com/, 2013.

[51] "Firefox boot2gecko project: Webapi." https://wiki.mozilla.org/WebAPI,

2013.

[52] "How To Determine Application Capabilities." http://msdn.microsoft.com/en-us/library/gg180730(v=vs.92).aspx, April 2012.

[53] Aaron, K., "The smartphone safety tip you need." http://www.prevention.com/health/healthy-living/smartphone-users-vulnerable-security-threats, 2012.

[54] Adida, B., "Beamauth: two-factor web authentication with a bookmark," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2007.

[55] Aggarwal, G., Bursztein, E., Jackson, C., and Boneh, D., "An Analysis of Private Browsing Modes in Modern Browsers," in *USENIX Security Symposium*, 2010.

[56] Amrutkar, C., Hiltunen, M., Jim, T., Joshi, K., Spatscheck, O., Traynor, P., and Venkataraman, S., "Why is my smartphone slow? on the fly diagnosis of underperform ance on the mobile internet," in *Proceedings of the 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2013.

[57] Amrutkar, C., Singh, K., and Traynor, P., "On the Disparity of Display Security in Mobile and Traditional Web Browsers," tech. rep., GT-CS-11-02, Georgia Institute of Technology, 2011.

[58] Amrutkar, C., Singh, K., Verma, A., and Traynor, P., "VulnerableMe: Measuring systemic weaknesses in mobile browser security," in *Proceedings of the International Conference on Information Systems Security (ICISS)*, 2012.

[59] Amrutkar, C. and Traynor, P., "Is browsing internet on your mobile phone secure? (an evaluation of display security in mobile and traditional web browsers)." USENIX Security Symposium (SECURITY) Poster Session, 2011.

[60] Amrutkar, C. and Traynor, P., "Rethinking permissions for mobile web

apps: Barriers and the road ahead," in *Proceedings of ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices (CCS-SPSM)*, 2012.

[61] Amrutkar, C., Traynor, P., and van Oorschot, P. C., "Measuring SSL indicators on mobile browsers: Extended life, or end of the road?," in *Proceedings of the Information Security Conference (ISC)*, 2012.

[62] Amrutkar, C., van Oorschot, P. C., and Traynor, P., "An Empirical Evaluation of Security Indicators in Mobile Web Browsers." Georgia Tech Technical Report GT-CS-11-10, 2011.

[63] Antonakakis, M., Perdisci, R., Dagon, D., Lee, W., and Feamster, N., "Building a dynamic reputation system for DNS," in *Proceedings of the 19th USENIX Conference on Security (SECURITY)*, 2010.

[64] Balasubramaniyan, V. A., Poonawalla, A., Ahamad, M., Hunter, M. T., and Traynor, P., "Pindr0p: using single-ended audio features to determine call provenance," in *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS)*, 2010.

[65] Bandhakavi, S., King, S. T., Madhusudan, P., and Winslett, M., "VEX: Vetting Browser Extensions For Security Vulnerabilities," in *Proceedings of the USENIX Security Symposium (SECURITY)*, 2010.

[66] Barrera, D., Kayacik, H. G., van Oorschot, P. C., and Somayaji, A., "A methodology for empirical analysis of permission-based security models and its application to android," in *Proceedings of the 17th ACM conference on Computer and communications security (CCS)*, 2010.

[67] Barth, A., "The Web Origin Concept." http://tools.ietf.org/html/rfc6454.

[68] Barth, A., Caballero, J., and Song, D., "Secure Content Sniffing for Web Browsers, or How to Stop Papers from Reviewing Themselves," in *Proceedings of the IEEE Symposium on Security and Privacy, Oakland*, 2009.

[69] Barth, A., Felt, A. P., Saxena, P., and Boodman, A., "Protecting

Browsers from Extension Vulnerabilities," in *Proceedings of the 17th Network and Distributed System Security Symposium (NDSS)*, 2010.

[70] Barth, A. and Jackson, C., "Protecting Browsers from Frame Hijacking Attacks." `http://seclab.stanford.edu/websec/frames/navigation/`.

[71] Barth, A., Jackson, C., and Mitchell, J. C., "Robust Defenses for Cross-Site Request Forgery," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2008.

[72] Barth, A., Jackson, C., and Mitchell, J. C., "Securing frame communication in browsers," in *Proceedings of the USENIX Security Symposium (SECURITY)*, 2008.

[73] Barth, A., Jackson, C., Reis, C., and The Google Chrome Team, "The security architecture of the chromium browser." `http://seclab.stanford.edu/websec/chromium/chromium-security-architecture.pdf`.

[74] Barth, A., Weinberger, J., and Song, D., "Cross-origin javascript capability leaks: detection, exploitation, and defense," in *Proceedings of the USENIX Security Symposium (SECURITY)*, 2009.

[75] Bernstein, D. J., "The qmail security guarantee." `http://cr.yp.to/qmail/guarantee.html`.

[76] Biddle, R., Van Oorschot, P., Patrick, A., Sobey, J., and Whalen, T., "Browser interfaces and extended validation SSL certificates: an empirical study," in *Proceedings of the ACM workshop on Cloud computing security*, 2009.

[77] Bilge, L., Kirda, E., Kruegel, C., and Balduzzi, M., "EXPOSURE : Finding malicious domains using passive DNS analysis," in *Proceedings of the 18th Annual Network and Distributed System Security Symposium (NDSS)*, 2011.

[78] Bittau, A., Marchenko, P., Handley, M., and Karp, B., "Wedge: splitting applications into reduced-privilege compartments," in *Proceedings of the*

5th USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2008.

[79] Blog, G. M. A., "Smartphone user study shows mobile movement under way." `http://googlemobileads.blogspot.com/2011/04/smartphone-user-study-shows-mobile.html`, 2011.

[80] Boodaei, M., "Mobile users three times more vulnerable to phishing attacks." `http://www.trusteer.com/blog/mobile-users-three-times-more-vulnerable- phishing-attacks`, 2011.

[81] Boodaei, M., "Mobile users three times more vulnerable to phishing attacks." `http://www.trusteer.com/blog/mobile-users-three-times-more-vulnerable-to-phishing-attacks`, 2011.

[82] Budiu, R., "Mobile: Native apps, web apps, and hybrid apps." `http://www.nngroup.com/articles/mobile-native-apps/`.

[83] Butkiewicz, M., Wu, Z., Li, S., Murali, P., Hristidis, V., Madhyastha, H. V., and Sekar, V., "Enabling the transition to the mobile web with websieve," in *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications (HotMobile)*, 2013.

[84] Canali, D., Cova, M., Vigna, G., and Kruegel, C., "Prophiler: a fast filter for the large-scale detection of malicious web pages," in *Proceedings of the 20th International Conference on World Wide Web (WWW)*, 2011.

[85] Carlini, N., Felt, A. P., and Wagner, D., "An evaluation of the google chrome extension security architecture," in *Proceedings of the 21st USENIX conference on Security symposium*, 2012.

[86] Chakradeo, S., Reaves, B., Traynor, P., and Enck, W., "MAST: Triage for Market-scale Mobile Malware Analysis," Tech. Rep. GT-CS-12-01, College of Computing, Georgia Institute of Technology, 2012.

[87] Chakradeo, S., Reaves, B., Traynor, P., and Enck, W., "Mast: triage

for market-scale mobile malware analysis," in *Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks (WiSec)*, 2013.

[88] Charles Arthur, "Mobile internet devices 'will outnumber humans this year'." http://www.theguardian.com/technology/2013/feb/07/mobile-internet-outnumber-people.

[89] Chin, E., Felt, A. P., Greenwood, K., and Wagner, D., "Analyzing inter-application communication in Android," in *Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2011.

[90] Choney, S., "Mobile browser woes can fool even experts: report." http://www.nbcnews.com/technology/mobile-browser-woes-can-fool-even-experts-report-1C7451203, 2012.

[91] Chou, N., Ledesma, R., Teraguchi, Y., Boneh, D., and Mitchell, J., "Client-side defense against web-based identity theft," in *Proc. NDSS*, 2004.

[92] Cova, M., Kruegel, C., and Vigna, G., "Detection and analysis of drive-by-download attacks and malicious javascript code," in *Proceedings of the 19th international conference on World Wide Web (WWW)*, 2010.

[93] Cuervo, E., Balasubramanian, A., Cho, D.-k., Wolman, A., Saroiu, S., Chandra, R., and Bahl, P., "Maui: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services (MobiSys)*, 2010.

[94] Curtsinger, C., Livshits, B., Zorn, B., and Seifert, C., "Zozzle: fast and precise in-browser javascript malware detection," in *Proceedings of the 20th USENIX conference on Security (SECURITY)*, 2011.

[95] Davies, C., "iPhone Os Safari Vulnerable To DoS Attacks." http://www.iphonebuzz.com/iphone-safari-dos-bug-discovered-162212.php, April

16 2008.

[96] Desmet, L., Joosen, W., Massacci, F., Piessens, F., Siahaan, I., and Vanoverberghe, D., "Security by contract on the .NET platform," tech. rep., Information Security Technical 13, January 2008.

[97] Dhamija, R., Tygar, J. D., and Hearst, M., "Why phishing works," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2006.

[98] Dhamija, R. and Tygar, J., "The battle against phishing: Dynamic security skins," in *Proceedings of the symposium on Usable privacy and security*, 2005.

[99] Doug Seven, "What is a Hybrid Mobile App?." `http://www.icenium.com/blog/icenium-team-blog/2012/06/14/what-is-a-hybrid-mobile-app-`.

[100] Downs, J., Holbrook, M., and Cranor, L., "Decision strategies and susceptibility to phishing," in *Proceedings of the Second Symposium on Usable Privacy and Security*, 2006.

[101] Drake, D., Mehta, P., Miller, C., Moyer, S., Smith, R., and Valasek, C., " Browser Security Comparison: A Quantitative Approach," tech. rep., Accuvant labs, 2011.

[102] Egele, M., Kruegel, C., Kirda, E., and Vigna, G., "PiOS: Detecting Privacy Leaks in iOS Applications," in *Proceedings of the ISOC Networking & Distributed Systems Security (NDSS) Symposium*, 2011.

[103] Egelman, S., Cranor, L. F., and Hong, J., "You've been warned: an empirical study of the effectiveness of web browser phishing warnings," in *Proceedings of the 26th annual SIGCHI conference on Human factors in computing systems*, 2008.

[104] ELR, G. N., "Hackers steal players' passwords of 'league of legends' online game." `http://www.gmanetwork.com/news/story/261386/scitech/gaming/hackers-steal-players-passwords-of-league-of-legends-online-game`,

2012.

[105] Enck, W., Gilbert, P., Chun, B.-G., Cox, L. P., Jung, J., McDaniel, P., and Sheth, A. N., "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones," in *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2010.

[106] Enck, W., Octeau, D., McDaniel, P., and Chaudhuri, S., "A study of android application security," in *Proceedings of the 20th USENIX conference on Security*, 2011.

[107] Enck, W., Ongtang, M., and McDaniel, P., "On Lightweight Mobile Phone Application Certification," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2009.

[108] Feinstein, B. and Peck, D., "Caffeine monkey: Automated collection, detection and analysis of malicious javascript," in *Proceedings of the Black Hat Security Conference*, 2007.

[109] Felt, A. P., Chin, E., Hanna, S., Song, D., and Wagner, D., "Android permissions demystified," in *Proceedings of the 18th ACM conference on Computer and communications security*, 2011.

[110] Felt, A. P., Finifter, M., Weinberger, J., and Wagner, D., "Diesel: applying privilege separation to database access," in *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, 2011.

[111] Felt, A. P., Greenwood, K., and Wagner, D., "The effectiveness of application permissions," in *Proceedings of the 2nd USENIX conference on Web application development*, 2011.

[112] Felt, A. P. and Wagner, D., "Phishing on mobile devices," in *Web 2.0 Security and Privacy (W2SP)*, 2011.

[113] Felten, E. W., Balfanz, D., Dean, D., and Wallach, D. S., "Intrusion

Detection Prevention Web Spoofing: An Internet Con Game," in *20th National Information Systems Security Conference*, 1997.

[114] Fette, I., Sadeh, N., and Tomasic, A., "Learning to detect phishing emails," in *Proceedings of the 16th International Conference on World Wide Web (WWW)*, 2007.

[115] Friedman, B., Hurley, D., Howe, D., Felten, E., and Nissenbaum, H., "Users' conceptions of web security: a comparative study," in *CHI extended abstracts on Human factors in computing systems*, 2002.

[116] Gajek, S., Sadeghi, A.-R., Stüble, C., and Winandy, M., "Compartmented security for browsers – or how to thwart a phisher with trusted computing," in *Second International Conference on Availability, Reliability and Security (ARES)*, 2007.

[117] Garera, S., Provos, N., Chew, M., and Rubin, A. D., "A framework for detection and measurement of phishing attacks," in *Proceedings of the ACM workshop on recurring malcode*, 2007.

[118] Gibler, C., Crussell, J., Erickson, J., and Chen, H., "Androidleaks: automatically detecting potential privacy leaks in android applications on a large scale," in *Proceedings of the 5th international conference on Trust and Trustworthy Computing (TRUST)*, 2012.

[119] Gold, J., "Ga. tech researchers: Mobile browsers need better https indicators." `http://www.networkworld.com/news/2012/120512-mobile-browsers-264846.html`, 2012.

[120] Google Caja Team, "Google-Caja: A source-to-source translator for securing JavaScript based Web." `http://code.google.com/p/google-caja/`.

[121] Grace, M., Zhou, Yajin Wang, Z., and Jiang, X., "Systematic Detection of Capability Leaks in Stock Android Smartphones," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2012.

[122] Grier, C., King, S. T., and Wallach, D. S., "How I Learned to Stop Worrying and Love Plugins," in *In Web 2.0 Security and Privacy*, 2009.

[123] Grier, C., Tang, S., and King, S. T., "Secure Web Browsing with the OP Web Browser," in *Proceedings of the IEEE Symposium on Security and Privacy (Oakland)*, 2008.

[124] Guha, A., Fredrikson, M., Livshits, B., and Swamy, N., "Verified security for browser extensions," in *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, 2011.

[125] Gus Andrews, "Has the address bar had its day?." `http://www.netmagazine.com/features/has-address-bar-had-its-day`.

[126] Herzberg, A. and Jbara, A., "Security and identification indicators for browsers against spoofing and phishing attacks," *ACM Transactions on Internet Technology*, 2008.

[127] Holz, T., Gorecki, C., Rieck, K., and Freiling, F. C., "Measuring and detecting fast-flux service networks," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2008.

[128] HTML5 contributors, "The Web platform: Browser technologies." `http://platform.html5.org/`.

[129] Huang, L.-S., Weinberg, Z., Evans, C., and Jackson, C., "Protecting browsers from cross-origin CSS attacks," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2010.

[130] Ikinci, A., Holz, T., and Freiling, F., "Monkey-spider: Detecting malicious websites with low-interaction honeyclients," in *Proceedings of Sicherheit, Schutz und Zuverlassigkeit*, 2008.

[131] Invernizzi, L., Benvenuti, S., Cova, M., Comparetti, P. M., Kruegel, C., and Vigna, G., "Evilseed: A guided approach to finding malicious web pages," in *Proceedings of the 2012 IEEE Symposium on Security and*

Privacy, 2012.

[132] J. Nielsen, *Usability Engineering*. Morgan Kaufmann, 1993.

[133] Jackson, C. and Barth, A., "Beware of Finer-Grained Origins," in *Proceedings of the Workshop on Web 2.0 Security and Privacy (W2SP)*, 2008.

[134] Jackson, C., Bortz, A., Boneh, D., and Mitchell, J. C., "Protecting browser state from web privacy attacks," in *Proceedings of the 15th international conference on World Wide Web (WWW)*, 2006.

[135] Jackson, C., Bortz, A., Boneh, D., and Mitchell, J. C., "Alice in warningland: A large-scale field study of browser security warning effectiveness," in *Proceedings of the USENIX conference on Security*, 2013.

[136] Jackson, C., Simon, D., and Tan, D., "An evaluation of extended validation and picture-in-picture phishing attacks," *Financial Cryptography and Data*, 2007.

[137] Karim, R., Dhawan, M., Ganapathy, V., and Shan, C.-c., "An analysis of the mozilla jetpack extension framework," in *Proceedings of the 26th European conference on Object-Oriented Programming*, 2012.

[138] Kolari, P., Finin, T., and Joshi, A., "Svms for the blogosphere: Blog identification and splog detection," in *Proceedings of AAAI Spring Symposium on Computational Approaches to Analysing Weblogs*, 2006.

[139] Kolbitsch, C., Livshits, B., Zorn, B., and Seifert, C., "Rozzle: Decloaking internet malware," in *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, 2012.

[140] Konte, M., Feamster, N., and Jung, J., "Dynamics of online scam hosting infrastructure," in *Proceedings of the 10th International Conference on Passive and Active Network Measurement (PAM)*, 2009.

[141] Kovacs, E., "Chinese Hackers Develop Automated Tools to Exploit Apache Struts Vulnerabilities." `http://news.softpedia.com/news/`

Chinese-Hackers-Develop-Automated-Tools-to-Exploit-Apache-Struts-Vulnerabili
shtml, 2013.

[142] Krishnamurthy, A., Mettler, A., and Wagner, D., "Fine-grained privilege separation for web applications," in *Proceedings of the 19th international conference on World wide web (WWW)*, 2010.

[143] Krohn, M., Efstathopoulos, P., Frey, C., Kaashoek, F., Kohler, E., Mazières, D., Morris, R., Osborne, M., VanDeBogart, S., and Ziegler, D., "Make least privilege a right (not a privilege)," in *Proceedings of the 10th conference on Hot Topics in Operating Systems - Volume 10*, 2005.

[144] Le, A., Markopoulou, A., and Faloutsos, M., "Phishdef: Url names say it all," in *Proceedings of IEEE International Conference on Computer Communications (INFOCOM)*, 2011.

[145] Lekies, S. and Johns, M., "Lightweight integrity protection for web storage-driven content caching," in *In IEEE Oakland Web 2.0 Security and Privacy (W2SP)*, 2012.

[146] Lever, C., Antonakakis, M., Reaves, B., Traynor, P., and Lee, W., "The core of the matter: Analyzing malicious traffic in cellular carriers," in *Proceedings of Network and Distributed System Security Symposium (NDSS)*, 2013.

[147] Likarish, P., Jung, E., and Jo, I., "Obfuscated malicious javascript detection using classification techniques," in *Proceedings of Malicious and Unwanted Software (MALWARE)*, 2009.

[148] Liu, L., Zhang, X., Yan, G., and Chen, S., "Chrome Extensions: Threat Analysis and Countermeasures," in *Proceedings of the 17th Network and Distributed System Security Symposium (NDSS)*, 2012.

[149] Liverani, R. S. and Freeman, N., "Abusing firefox extensions." Defcon17, 2009.

[150] Livshits, B. and Molnar, D., "Empowering Browser Security for Mobile Devices Using Smart CDNs," in *Proceedings of the Workshop on Web 2.0 Security and Privacy (W2SP)*, 2010.

[151] Ludl, C., Mcallister, S., Kirda, E., and Kruegel, C., "On the effectiveness of techniques to detect phishing sites," in *Proceedings of the 4th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, 2007.

[152] Luttrell, M., "Majority of users prefer mobile browser over apps." `http://www.tgdaily.com/mobility-brief/55884-majority-of-users-prefer-mobile-browser-over-apps`, 2011.

[153] Ma, J., Saul, L. K., Savage, S., and Voelker, G. M., "Beyond blacklists: Learning to detect malicious web sites from suspicious URLs," in *Proceedings of the SIGKDD Conference*, 2009.

[154] Marlinspike, M., "More Tricks For Defeating SSL In Practice." `http://www.blackhat.com/presentations/bh-usa-09/MARLINSPIKE/BHUSA09-Marlinspike-DefeatSSL-SLIDES.pdf`, 2009.

[155] McGrath, D. K. and Gupta, M., "Behind phishing: an examination of phisher modi operandi," in *Proceedings of the 1st USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2008.

[156] Medvet, E., Kirda, E., and Kruegel, C., "Visual-similarity-based phishing detection," in *Proceedings of International Conference on Security and Privacy in Communication Netowrks (SecureComm)*, 2008.

[157] min Wang, Y., Beck, D., Jiang, X., Roussev, R., Verbowski, C., Chen, S., and King, S., "Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities," in *Proceedings of the Networking and Distributed Systems Security (NDSS)*, 2006.

[158] Moore, T., Clayton, R., and Stern, H., "Temporal correlations between

spam and phishing websites," in *Proceedings of the 2nd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more (LEET)*, 2009.

[159] Moshchuk, A., Bragin, T., Gribble, S. D., and Levy, H. M., "A crawler-based study of spyware on the web," in *Proceedings of Network and Distributed System Security Symposium (NDSS)*, 2006.

[160] Mulliner, C., Vigna, G., Dagon, D., and Lee, W., "Using labeling to prevent cross-service attacks against smart phones," in *Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, 2006.

[161] Murray, D. G. and Hand, S., "Privilege separation made easy: trusting small libraries not big processes," in *Proceedings of the 1st European Workshop on System Security (EUROSEC)*, 2008.

[162] Nadkarni, A. and Enck, W., "Preventing Accidental Data Disclosure in Modern Operating Systems," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2013.

[163] Nazario, J., "Phoneyc: a virtual client honeypot," in *Proceedings of the 2nd USENIX conference on Large-scale Exploits and Emergent Threats: botnets, spyware, worms, and more (LEET)*, 2009.

[164] Nazario, J. and Holz, T., "As the net churns: Fast-flux botnet observations," in *Proceedings of 3rd International Conference on Malicious and Unwanted Software (MALWARE)*, 2008.

[165] Niu, Y., Hsu, F., and Chen, H., "iPhish: Phishing Vulnerabilities on Consumer Electronics," in *Usability, Psychology, and Security*, 2008.

[166] O'Dell, J., "New study shows the mobile web will rule by 2015 [stats]." `http://mashable.com/2010/04/13/mobile-web-stats/`, 2010.

[167] Ongtang, M., McLaughlin, S., Enck, W., and McDaniel, P., "Semantically rich application-centric security in android," in *Annual Computer Security Applications Conference (ACSAC)*, 2009.

[168] OWASP, "SQL injection." https://www.owasp.org/index.php/SQL_Injection.

[169] OWASP, "OWASP top ten, the ten most critical web application security risks." http://owasptop10.googlecode.com/files/OWASP

[170] Pandita, R., Xiao, X., Yang, W., Enck, W., , and Xie, T., "Whyper: Towards automating risk assessment of mobile applications," in *Proceedings of the USENIX Security Symposium*, 2013.

[171] Passerini, E., Paleari, R., Martignoni, L., and Bruschi, D., "Fluxor: Detecting and monitoring fast-flux service networks," in *Proceedings of the 5th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, 2008.

[172] Payne, M., "Online power scammers steal credit card numbers." `http://www.news-press.com/article/20130827/COLUMNISTS40/130827023/Online-power-scammers-steal-credit-card-numbers`, 2013.

[173] Perdisci, R., Corona, I., Dagon, D., and Lee, W., "Detecting malicious flux service networks through passive analysis of recursive DNS traces," in *Proceedings of Annual Computer Security Applications Conference (ACSAC)*, 2009.

[174] Podjarny, G., "Mobile web performance optimization." `http://www.slideshare.net/blazeio/mobile-web-performance-optimization-tips-and-tricks`.

[175] Provos, N., Friedl, M., and Honeyman, P., "Preventing privilege escalation," in *Proceedings of the 12th conference on USENIX Security Symposium*, 2003.

[176] Provos, N., Mavrommatis, P., Rajab, M. A., and Monrose, F., "All your iframes point to us," in *Proceedings of the 17th USENIX conference on*

Security (SECURITY), 2008.

[177] Ratanaworabhan, P., Livshits, B., and Zorn, B., "Nozzle: a defense against heap-spraying code injection attacks," in *Proceedings of the 18th conference on USENIX security symposium*, 2009.

[178] Resig, J., "iPhone overflow clickjacking." `http://ejohn.org/blog/clickjacking-iphone-attack/`, November 2008.

[179] Rieck, K., Krueger, T., and Dewald, A., "Cujo: efficient detection and prevention of drive-by-download attacks," in *Proceedings of the 26th Annual Computer Security Applications Conference (ACSAC)*, 2010.

[180] Rodgers, L. and Nicewander, W. A., "Thirteen ways to look at the correlation coefficient," *The American Statistician*, 1988.

[181] Ruderman, J., "Same Origin Policy for JavaScript." `http://www.mozilla.org/projects/security/components/same-origin.html`.

[182] Rydstedt, G., Bursztein, E., Boneh, D., and Jackson, C., "Busting Frame Busting: A Study of Clickjacking Vulnerabilities at Popular Sites," in *Proceedings of the IEEE Web 2.0 Security and Privacy Workshop (W2SP)*, 2010.

[183] Rydstedt, G., Gourdin, B., Bursztein, E., and Boneh, D., "Framing Attacks on Smart Phones and Dumb Routers: Tap-jacking and Geo-localization Attacks," in *Proceedings of the USENIX Workshop on Offensive Technology (WOOT)*, 2010.

[184] Saltzer, J. and Schroeder, M. D., "The protection of information in computer systems." In IEEE 63, 1975.

[185] Schechter, S., Dhamija, R., Ozment, A., and Fischer, I., "The Emperor's New Security Indicators," in *Proceedings of IEEE Symposium on Security and Privacy*, 2007.

[186] Schwartz, M. J., "Blame screen size: Mobile browsers flunk security tests." `http://www.informationweek.com/security/mobile/`

blame-screen-size-mobile-browsers-flunk/240143999, 2012.

[187] Security News Portal, "Symbian – Just Because it's Signed Doesn't Mean it Isn't Spying on You." http://www.securitynewsportal.com/securitynews/article.php?TITLE=Just_because_its_Signed_doesnt_mean_it_isnt_spying_on_you, 2007.

[188] Seifert, C., Welch, I., and Komisarczuk, P., "Identification of malicious web pages with static heuristics," in *Telecommunication Networks and Applications Conference*, 2008.

[189] Singh, K., "Can Mobile learn from the Web?," in *Proceedings of the Workshop on Web 2.0 Security and Privacy (W2SP)*, 2012.

[190] Singh, K., "Practical context-aware permission control for hybrid mobile applications," in *16th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, 2013.

[191] Singh, K., Moshchuk, A., Wang, H. J., and Lee, W., "On the Incoherencies in Web Browser Access Control Policies," in *IEEE Symposium on Security and Privacy (Oakland)*, 2010.

[192] Sobey, J., Biddle, R., van Oorschot, P., and Patrick, A., "Exploring user reactions to new browser cues for extended validation certificates," in *European Symposium on Research in Computer Security (ESORICS)*, 2008.

[193] Stebila, D., "Reinforcing bad behaviour: the misuse of security indicators on popular websites," in *Proceedings of the 22nd Conference of the Computer-Human Interaction Special Interest Group of Australia on Computer-Human Interaction*, 2010.

[194] Sterne, B. and Barth, A., "Content Security Policy 1.1." https://dvcs.w3.org/hg/content-security-policy/raw-file/tip/csp-specification.dev.html.

[195] Stewart, D. and Martin, I., "Intended and Unintended Consequences of

Warning Messages: A Review and Synthesis of Empirical Research," 1994.

[196] Sunshine, J., Egelman, S., Almuhimedi, H., Atri, N., and Cranor, L. F., "Crying Wolf: An Empirical Study of SSL Warning Effectiveness," in *Proceedings of 18th USENIX Security Symposium (SECURITY)*, 2009.

[197] Symantec, "Symantec internet security threat report." `http://eval.symantec.com/mktginfo/enterprise/white_papers/b-whitepaper_exec_summary_internet_security_threat_report_xiii_04-2008.en-us.pdf`, 2007.

[198] Tang, S., Mai, H., and King, S. T., "Trust and protection in the Illinois browser operating system," in *Proceedings of the USENIX Conference on Operating Systems Design and Implementation (OSDI)*, 2010.

[199] Terrazas, M., "Mobile browsers fail georgia tech safety test." `http://technews.acm.org/archives.cfm?fo=2012-12-dec/dec-07-2012.html#622306`, 2012.

[200] The Open Mobile Alliance, "Wireless Application Protocol (WAP) 1.0 Specification Suite." `http://www.wapforum.org/what/technical_1_0.htm`, 1998.

[201] Thiagarajan, N., Aggarwal, G., Nicoara, A., Boneh, D., and Singh, J. P., "Who killed my battery?: analyzing mobile browser energy consumption," in *Proceedings of the 21st international conference on World Wide Web (WWW)*, 2012.

[202] Traynor, P., Amrutkar, C., Rao, V., Jaeger, T., McDaniel, P., and Porta, T. L., " From Mobile Phones to Responsible Devices," *Journal on Security and Communications Networks (SCN)*, vol. 4, pp. 719–726, June 2011.

[203] Traynor, P., Lin, M., Ongtang, M., Rao, V., Jaeger, T., La Porta,

T., and McDaniel, P., "On Cellular Botnets: Measuring the Impact of Malicious Devices on a Cellular Network Core," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2009.

[204] Vidas, T., Christin, N., and Cranor, L. F., "Curbing android permission creep," in *Proceedings of the Workshop on Web 2.0 Security and Privacy (W2SP)*, 2011.

[205] Villamarín-Salomón, R. and Brustoloni, J. C., "Bayesian bot detection based on dns traffic similarity," in *Proceedings of the 2009 ACM symposium on Applied Computing (SAC)*, 2009.

[206] Vratonjic, N., Freudiger, J., Bindschaedler, V., and Hubaux, J.-P., "The inconvenient truth about web certificates," in *The Workshop on Economics of Information Security (WEIS)*, 2011.

[207] W3C, "Web storage." `http://dev.w3.org/html5/webstorage`.

[208] Wang, H. J., Fan, X., Howell, J., and Jackson, C., "Protection and communication abstractions for web browsers in MashupOS," in *Proceedings of 21st ACM SIGOPS symposium on Operating systems principles*, 2007.

[209] Wang, H. J., Grier, C., Moshchuk, A., King, S. T., Choudary, P., and Venter, H., "The Multi-Principal OS Construction of the Gazelle Web Browser," in *Proceedings of the USENIX Security Symposium (SECURITY)*, 2009.

[210] Weimer, F., "Passive DNS replication," 2005.

[211] Whalen, T. and Inkpen, K., "Gathering evidence: use of visual security cues in web browsers," in *Proceedings of Graphics Interface*, 2005.

[212] Whittaker, C., Ryner, B., and Nazif, M., "Large-scale automatic classification of phishing pages," in *Proceedings of the Networking and Distributed Systems Security (NDSS)*, 2010.

[213] Xiang, G., Hong, J., Rose, C. P., and Cranor, L., "Cantina+: A feature-rich machine learning framework for detecting phishing web sites," *ACM Transactions on Information and System Security (TISSEC)*, vol. 14, Sept. 2011.

[214] Ye, Z. E., Smith, S., and Anthony, D., "Trusted paths for browsers," *ACM Transactions on Information and System Security (TISSEC)*, May 2005.

[215] Yue, C. and Wang, H., "Characterizing insecure javascript practices on the web," in *Proceedings of the 18th international conference on World Wide Web (WWW)*, 2009.

[216] Zalewski, M., "Browser Security Handbook." `https://code.google.com/p/browsersec/wiki/Main`.

[217] Zdrnja, B., Brownlee, N., and Wessels, D., "Passive monitoring of DNS anomalies," in *Proceedings of the 4th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, 2007.

[218] Zhang, Y., Hong, J. I., and Cranor, L. F., "Cantina: a content-based approach to detecting phishing web sites," in *Proceedings of the 16th international conference on World Wide Web (WWW)*, 2007.

[219] Zhou, W., Zhou, Y., Jiang, X., and Ning, P., "DroidMOSS: Detecting Repackaged Smartphone Applications in Third-Party Android Marketplaces," in *Proceedings of the 2nd ACM Conference on Data and Application Security and Privacy*, 2012.

[220] Zhou, Y., Wang, Z., Zhou, W., and Jiang, X., "Hey, You, Get off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets," in *Proceedings of the Network and Distributed System Security Symposium*, 2012.

[221] Zhou, Y. and Evans, D., "Why Aren't HTTP-only Cookies More Widely Deployed?," in *Proceedings of the IEEE Web 2.0 Security and Privacy Workshop (W2SP)*, 2010.