On the Disparity of Display Security in Mobile and Traditional Web Browsers

Chaitrali Amrutkar, Kapil Singh, Arunabh Verma and Patrick Traynor

Converging Infrastructure Security (CISEC) Laboratory
Georgia Institute of Technology

Abstract

Mobile web browsers now provide nearly equivalent features when compared to their desktop counterparts. However, smaller screen size and optimized features for constrained hardware make the web experience on mobile browsers significantly different. In this paper, we present the first comprehensive study of the display-related security issues in mobile browsers. We identify two new classes of display-related security problems in mobile browsers and devise a range of real world attacks against them. Additionally, we identify an existing security policy for display on desktop browsers that is inappropriate on mobile browsers. Our analysis is comprised of *eight* mobile and *five* desktop browsers. We compare security policies for display in the candidate browsers to infer that desktop browsers are significantly more compliant with the policies as compared to mobile browsers. We conclude that mobile browsers create new security challenges and are not simply miniature versions of their desktop counterparts.

1 Introduction

Mobile web browsers have long underperformed their desktop counterparts. Whether by implementing limited alternative standards such as WAP [34] 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 are now able to render complex webpages that contain many popular embedded elements (e.g., Javascript). The most popular browsers, including iPhone Safari, Android, Opera Mini and Blackberry (v6.0+), even rely on the same rendering engines [15, 14] used by many desktop browsers. Accordingly, screen size is now arguably one of the most distinguishable differences between mobile and desktop platforms. However, this difference is more than cosmetic. Specifically, there exists a fundamental tension between a user's ability to see large sections of a webpage and concurrently interact with any of its individual elements. The difficulty in efficiently accessing large pages on mobile devices makes an adversary capable of abusing the rendering of display elements particularly acute from a security perspective.

In this paper, we characterize a number of differences in the ways mobile and desktop browsers render webpages that potentially allow an adversary to deceive mobile users into performing unwanted and potentially dangerous operations. Specifically, we examine the handling of user interaction with overlapped display elements, the ability of malicious cross-origin elements to affect the placement of honest elements and the ability of malicious cross-origin elements to alter the navigation of honest parent and descendant elements. We then perform the same tests against a number of desktop browsers and find that the majority of desktop browsers are not vulnerable to the same rendering issues. We show that such attacks are due to the lack of enforcement of display security policies and exacerbated by screen size limitations, making many such attacks difficult for users to detect.

This paper makes the following contributions:

- **Propose a new security policy for event handling for overlapped elements:** Browsers currently make only one access decision when the pixels of two elements overlap - which element owns the visual property of a pixel (display). We argue that browsers must also enforce which element should then respond to user access to that pixel and define a policy reflecting this decision. We then show that failure to enforce this policy renders browsers susceptible to a range of attacks including login CSRF and click fraud.

- **Evaluate pertinence and enforcement of previously proposed display security policies:** Recent work in this space has resulted in the proposal and implementation of a number of security policies for display elements in the desktop space. We evaluate the relevance of navigation policies [10] for desktop browsers in mobile browsers, demonstrate a widespread lack of enforcement of policies regarding display access [36] and demonstrate that such oversights expose mobile

browsers to attacks including phishing and password stealing.

- **Perform tests against a wide-range of the most popular mobile and desktop browsers:** We explore compliance with display security policies on *eight* mobile (Android Mobile, Blackberry (Mango), Blackberry (Webkit), Firefox Mobile, Nokia Mini-Map, Opera Mini, iPhone Safari and Internet Explorer Mobile) and *five* desktop (Chrome, Firefox, Internet Explorer, Opera and Safari) browsers. Our experiments reveal a range of rendering vulnerabilities that potentially allow an adversary to force users to perform dangerous actions almost exclusively in mobile browsers.

The remainder of the paper is organized as follows: Section 2 discusses background information, our methodology and presents our threat model; Section 3 presents our policy regarding user access to overlapping elements and evaluates its enforcement in current browsers; Section 4 evaluates compliance with the display access policy as it relates to cross-origin elements; Section 5 examines the ability of cross-origin elements to influence navigation; Section 7 provides higher-level observations and potential mitigation costs; Section 8 provides context for this work in terms of related work; Section 9 offers concluding thoughts.

Browser Name	**% Market Share**
Opera	21.04
iPhone	18.17
Nokia	15.52
Blackberry	14.85
Android	14.19
iPod Touch	6.73
Other (Firefox Mobile, IE on Windows 7 phone OS etc.)	3.25

Table 1: Market Share of Popular Mobile Browsers as of January 2011. We cover approximately 90.5% of the mobile browsers in the market for our evaluation.

	Landlord	Tenant
position(x,y,z)	RW	
dimensions (height, width)	RW	R
pixels		RW
URL location	W	RW

Table 2: Access control policy for landlord and its cross-original tenant as defined by Wang et al [36]. Only the landlord controls the position and dimensions of the tenant. Only the tenant can read and write his own pixels. The landlord can overlay the tenant completely with other content.

2 Overview

In this section, we provide an overview of display elements and rendering techniques for mobile browsers. We then discuss our experimental methodology and define our threat model and potential adversaries.

2.1 Display Handling in Mobile Browsers

Content not fitting the window in early HTML-capable mobile browsers would either be wrapped or would not be displayed at all [2]. Modern mobile browsers adapt to such differences using a range of techniques. For many webpages, mobile browsers attempt to display as much as possible and scale content based on the width and height of the browser. This approach can make content on a webpage appear very small and unreadable. Users must then move to and zoom in to the area containing the desired content. To avoid user effort, a growing number of pages instead rely on the "viewport" metatag. This metatag allows web developers to control the size and initial zoom level of the content and also constrain the user's control of scaling the page. Mobile specific CSS files [12] and compression of webpage content for a phone screen [2] are other techniques used for handling small screen size.

Scaling the page to the device's dimensions or zooming may not necessarily provide the best user experience. Some websites instead design pages specific to mobile screensi which look considerably different from their desktop counterparts. Such pages contain limited content and often provide a subset of functionalities. For example, Facebook's mobile web page does not provide the chat feature found in the desktop version. Many mobile pages also offer the ability to navigate to the standard version of the webpage should the user prefer it.

Regardless of the method used to display content on a mobile browser, it is difficult to display large amounts of content and allow users to meaningfully interact with that content at the same time. Accordingly, poorly defined policies regarding how content is rendered by mobile browsers put users of these platforms at greater security risk than those of traditional desktop browsers.

2.2 Methodology

We analyze the rendering differences between popular desktop and mobile browsers for security. Existing desktop and mobile browsers do not have well defined and standardized security policies for display. Wang et al. [36] and Singh et al. [31] have previously studied display security in desktop browsers. In this work, we analyze mobile browsers and provide the first comprehensive study of their display properties. In particular, we propose a new policy to define proper handling of user access to overlapped pixels, evaluate previously proposed display access policies for display [36] and then study the window navigation policies that have been previously explored as part of display policies.

To verify the compliance of the security policies for display, we study *eight* mobile and *five* desktop browsers. Our study explores display security for approximately 90.5% of

Type	Browser Name	Version	Rendering Engine	Operating System	Device
Mobile	Opera Mini	5.5.1	Presto	Android 2.2.1	Nexus 1
	Safari	*	Webkit	iOS 4.1 (8B117)	iPhone
	Android	2.2.1	Webkit	Android 2.2.1	Nexus 1
	Blackberry	5.0.0	Mango	Blackberry OS 5.0.0.732	Bold 9650
	Blackberry	6.0.0	Webkit	Blackberry OS 6	Torch 9800
	Nokia Mini-Map	*	Webkit	Symbian S60	E71x
	Internet Explorer Mobile	*	Trident	Windows Phone 7.0.7004.0 OS	Samsung focus SGH-i917
	Firefox Mobile	4 Beta 3	Gecko	Android 2.2.1	Nexus 1
Desktop	Opera	11.00	Presto	OS X 10.6.5	–
	Safari	5.0.3	Webkit	OS X 10.6.5	–
	Chrome	8.0.552.237	Webkit	OS X 10.6.5	–
	Firefox	3.5.16	Gecko	OS X 10.6.5	–
	Internet Explorer	8.0.7600.16385	Trident	Windows 7	–

Table 3: Details of browsers used for experimental evaluation. We consider eight mobile and five desktop browsers. *: After thorough search for version numbers of these browsers, we could not find their specifications. We have used the default browsers shipped with the referenced version of the OS.

mobile browsers in the market [8]. Table 1 provides a breakdown of market share. Some of the browser vendors have platform specific browser versions - the Opera browser is available for Android, iPhone and a range of other platforms. In such cases, we choose one platform specific browser to perform our experiments. For example, we select the Opera Mini browser on an Android handset for our evaluation. Additionally, we evaluate only the iPhone Safari browser and argue that the iTouch Safari browser will have the same results as both devices run the same operating system and browser.

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`, `button` and `list` all fall under display elements. However, HTML elements such as `head`, `input` or `option` do not qualify as display elements. We create test cases consisting of display elements and study their security policies under different settings. For example, we study the security policies of cross-origin display elements: *when they overlap, when they access each other's display resources and when they are navigated to new sources*. We run each of our test cases on all the candidate browsers. We experimentally discover several vulnerabilities in popular mobile browsers due to non-compliance with security policies for display. We then describe potential attacks exploiting these vulnerabilities on mobile browsers. Additionally, we identify an existing display-related security policy on desktop browsers that is inappropriate on mobile browsers. Finally, we provide a comparison between mobile and desktop browsers for display-related security.

Our evaluation explores both new and previously proposed policies for display security. After proposing our own policy, we evaluate mobile browsers for compliance with the access control matrix for display as defined by Wang et al [36]. The matrix is shown for convenience in Table 2. The position attribute describes a display element's situation on the parent's page and dimensions define the area covered by the element. URL location is the path to the element's resource. Pixels refer to the actual private display content of the element. We extend the definition of pixels by including the user interaction with the tenant's pixels as part of private content of the tenant.

Table 3 provides details of each of the browsers used for evaluation. All our tests were performed on browsers on real mobile phones, and are recreated in the respective emulators to create many of the figures throughout the paper. In addition to the top *six* browsers in the market (Table 1), we also perform experiments on IE on Windows 7 phone OS and the Firefox Mobile browser on Android. Our evaluation was completed on January 14^{th} 2011.

2.3 Threat Model

We present three classes of adversaries. Each adversary tries to do one or more of the following: elevate his own privileges, attack other website principals or attack the user. Each adversary has knowledge of the security vulnerabilities in the display of mobile browsers. We describe the abilities and intention of each adversary and state the potential techniques that can be used by the adversary to attract victims to his content.

2.3.1 Landlord attacker

The *landlord attacker* is a malicious principal who can host his own websites such as *landlordattacker.com*. After hosting a malicious website, the landlord attacker hosts honest tenants on his website. 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 tries to exploit the honest tenant and/or

the honest user. The landlord has no control on the content of the tenant's rented area on the screen (Same Origin Policy). He 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. However, he cannot interfere with the content in the tenant's rented area. 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 *landlordattacker.com* will be exploited. Depending on the vulnerability that the landlord attacker is trying to exploit, 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 [21].

2.3.2 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 gadget in an honest website. Websites such as iGoogle allow any user having an account with them to upload a new gadget. Placing ads on honest websites is another way that the tenant attacker can insert malicious content in an honest website.

We assume that an honest user visits an honest website containing at least one tenant attacker area, on 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 on honest websites to attack the honest website and/or the user.

2.3.3 User attacker

The *user attacker* is a malicious user who wants to access restricted content on honest websites such as *honest.com*. The user has access to a range of mobile browsers and is knowledgeable of the display vulnerabilities in mobile browsers. His goal is to access restricted content on honest websites by exploiting these vulnerabilities.

The user attacker cannot tamper with the content and display of *honest.com*. He can only exploit the security vulnerabilities in mobile browsers to access unrestricted content on *honest.com*.

3 User Access to Overlapping Elements

When two or more elements share the same pixel on the screen, browsers must make two ownership decisions: which element owns the visual property (display) of the pixel and which element owns and responds to the user access to that pixel (user access). 'Display' belongs to the element that can manipulate the color of a pixel. We argue that in addition to following a security policy to constrain ownership of display of pixels, defining a security policy for 'user access to pixels' is equally crucial. We propose the following policy that addresses ownership of user access:

> **User Access to Overlapping Elements:** When two or more elements of any origin share the same pixel on the screen, the element on top should own and respond to the user access to the pixel, even if the top element is transparent.

This policy constrains the ownership of user access to overlapped pixels to a single principal. This behavior is different than the behavior of display ownership followed by popular desktop browsers (excluding IE 8 on Windows 7). The display of a pixel shared by overlapped elements, is owned by the first element on the top of a set of overlapping elements holding non-transparent data. If the top element does not render content at a pixel, the element immediately below it is given ownership of the display regardless of its origin. The first completely opaque element in a stack of overlapping elements becomes the final element sharing the display in the overlapped area. All the elements beneath the opaque element are not seen.

We argue that the 'user access' to overlapped pixels should not be shared and should be owned only by the top element's principal irrespective of its transparency. This policy should hold even if the overlapping elements are from the same-origin (Section 3.1.4). Our proposed policy for overlapping elements is currently the default behavior of the majority of the popular desktop browsers, although it has not previously been formally specified. While mobile browsers follow the ownership policy for display of pixels in overlapped area, we will show that most mobile browsers do not comply with the user access ownership policy and therefore are vulnerable to a range of attacks as follows. [1]

3.1 Attacks

We discuss potential attacks against browsers that do not comply with our proposed policy. Depending on the vulnerability in the browser, all three adversaries described in Section 2.3 can formulate attacks against non-compliant browsers.

3.1.1 Login CSRF

A browser not complying with the user access policy for overlapping elements is susceptible to the login CSRF (Cross Site Request Forgery) attack similar to one described by Barth et al. [20]. While previous work used DNS exploits

[1] Note that our proposed policy does not prevent *all possible* attacks (e.g. imitation attacks) due to overlapping elements. We discuss this in Section 3.3.

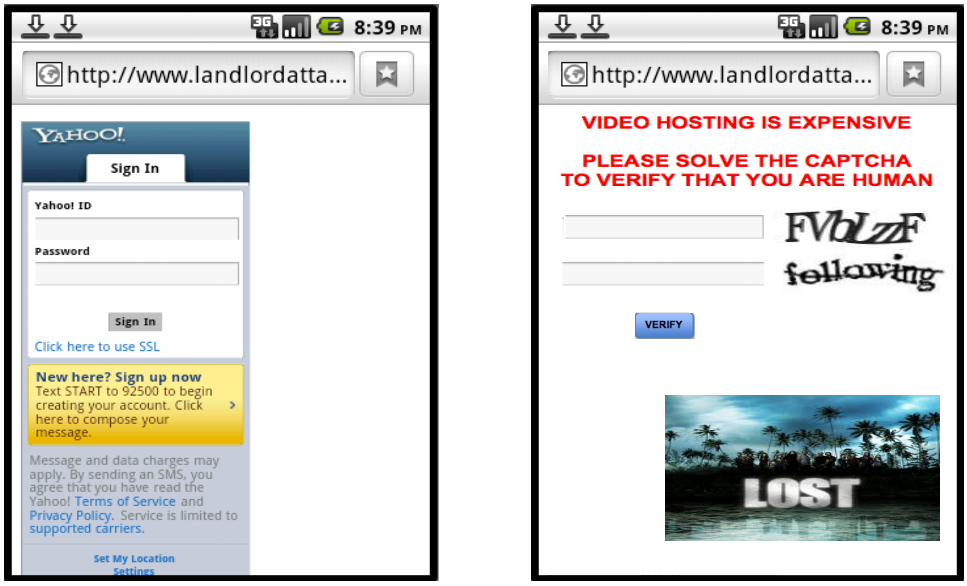

Figure 1: **Left image**: Login page of *www.yahoo.com* included in an iframe on *www.landlordattacker.com*; **Right image**: Image overlapping the *www.yahoo.com* iframe on *www.landlordattacker.com*. The text boxes for 'solving' the CAPTCHAs are placed exactly above the email and password fields on *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 google account. The Android (pictured), Opera Mini, Nokia and Firefox Mobile browsers are vulnerable to this attack.

to launch the attack, we show that a similar attack is feasible due to a browser's non-compliance to our proposed display policy. The intention of an attacker in a login CSRF is to make the honest user's browser log in *as the attacker* into a legitimate website without any notice to the user. A browser allowing user access to elements placed below an opaque display element is vulnerable to the login CSRF attack.

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 1. The landlord then overlaps the iframe completely with an opaque image as shown in Figure 1. The image shows enticing free content on the landlord's website and includes two 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. The landlord attacker then carefully places each of the solution boxes of the CAPTCHAs on the image exactly overlapping the email and password fields 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*, 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 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 gets clicked logging the user's browser into *www.yahoo.com* as the attacker.

Solving a CAPTCHA does not give out private user information and is perceived as a security feature of the website. 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 of solving the CAPTCHA.

Once the attacker is logged in from the user browser, all the consequences of login CSRF described by Barth et al. [20] are possible. The victim's search history on Yahoo search engine would be stored in the attacker's account after the attack is successful. Since Yahoo is a single sign-on website, all the other websites accessible by logging into *www.yahoo.com* once would be logged into as the attacker. For example, Yahoo offers a wide variety of websites such as shopping, finance, health etc. If the user browses to sensitive content such as health issues, his actions may be logged into the attacker's account.

3.1.2 Click Fraud

Any malicious principal (landlord or tenant) can execute a click fraud attack by exploiting non-compliance with the proposed user access policy as shown in Figure 2.

AdSense is one of Google's advertising programs. The advertisements that Google shows on a website are relevant to the website's content. A malicious principal can create an AdSense account free of charge and embed relevant content on his page to attract advertisements. The malicious principal earns money from Google when a user clicks on an

Figure 2: **Left image**: Image advertisement of sales in San Francisc on the *www.yelp.com* website ; **Right image**: The actual image placed below the enticing sales advertisement. A user clicks on the asbestos advertisements earn the attacker more money. The tenant on *www.yelp.com* is malicious and places the honest asbestos advertisement in an iframe and overlays it with the more enticing images of sales in San Francisco to increase the rate of clicks. Due to the vulnerability in the Opera Mini browser, when a user clicks on the sale advertisement in San Francisco, the asbestos advertisement is clicked benefiting the tenant attacker on Yelp. The Android, Opera Mini (pictured), iPhone Safari, Nokia and Firefox Mobile are vulnerable to the click fraud attack.

advertisement placed by Google AdSense. He cannot manipulate the advertisement placed by Google and thus cannot trick a user into clicking on an unwanted advertisement by disguising an AdSense advertisement with more enticing content.

If a browser allows a user to access content below an opaque element, the malicious principal can launch a click fraud attack. The attacker can include high paying advertisements such as advertisements for mesothelioma [1] and overlap each of them with more enticing content. If an honest user accesses the area containing the attractive content, the advertisements below would be clicked, benefiting the attacker.

3.1.3 Snooping

A malicious landlord can also learn about the interactions of the user with his cross-origin tenant by exploiting non-compliance with the proposed policy. If display elements situated below an opaque element respond to the user access instead of the element on top, a malicious landlord can launch a snooping attack. For example, consider a malicious page embedding an opaque cross-origin image advertisement with a click event. The expected behavior of `onclick` on the image is navigation of user's browser to the webpage of the advertiser. However, the malicious landlord can fill the entire screen area below the image with display elements whose `onclick` event implies user access to the advertisement in the image on top. Due to the incorrect handling of user access to overlapped pixels, if the user clicks on the image advertisement, the click event of the buttons below the image will be executed. This browser behavior will allow a malicious landlord to monitor the interaction of the user with the honest tenant breaching the access control policy in a landlord-tenant setting.

3.1.4 Escalation of User Privileges

The incorrect handling of user access ownership can allow a user adversary to elevate his privileges and access otherwise restricted data. For example, many free content hosting sites require the user to either fill out a survey or view an advertisement before accessing links to free online data. Web sites generally put a transparent iframe containing the survey on top of their webpage to avoid a user clicking on the links below. The transparency allows a user to view the content and the overlapped iframe is expected to block the user from clicking on the links. If a vulnerable browser allows a user to click through transparent iframes, the user adversary would simply render the webpage hosting the free content on that browser. If the bottom iframe containing the links to the free content also responds to the user access to overlapped pixels, the user adversary would be able to click on the links through the transparent iframe on top. Similar attack is possible when there is the bottom web page holds content out of the user's view. The user is expected to take the survey and then scroll to the hidden content. If the user is allowed to move to the otherwise hidden content by scrolling the bot-

tom iframe, he would be able to view the content restricted by the website. This will result in the malicious user easily bypassing the survey and breaking the expected behavior of the webpage. Even though the iframe containing the links to free content and the iframe containing the survey are from the same origin, it is important that only the top iframe responds to user access to the overlapped pixels.

3.2 Experimental Evaluation

We constructed a range of test cases to verify compliance with the proposed policy and ran them on the candidate browsers given in Section 2.2.

We observed that iframes in all the desktop and mobile browsers (except the IE desktop and mobile version), are transparent by default. Unless the developer colors an iframe, its "allowtransparency" property is set to true. We analyzed this default behavior of overlapping iframes in our candidate browsers. All desktop browsers except Safari follow the proposed policy correctly when iframes with the default transparency property overlap. In Safari, if two or more iframes of any origin overlap, a user is able to scroll all the contiguous transparent iframes from the top in the overlapped area. A user cannot scroll any iframe below the first opaque iframe from the top. This behavior may allow a user adversary to access otherwise restricted data in the bottom iframes by scrolling to the desired content leading to the escalation of user privileges attack.

Five (Android, Opera Mini, Nokia, Firefox Mobile and iPhone Safari) out of the *eight* mobile browsers do not correctly follow the proposed policy for overlapping elements. The browsers show different vulnerabilities leading to non-compliance of the proposed policy. Therefore, the potential attacks on each mobile browser vary. We will cover the most dangerous vulnerabilities that we discovered.

When transparent iframes overlap and share display pixels, the element owning the first colored pixel in the stack responds to the user access to the shared pixel in the Android and Opera Mini browsers. For example, a non-colored shared display pixel in the top iframe allows the user to access the first colored pixel immediately below it. The user can click buttons, write in text areas and also initiate events such as `onmouseover` and `onmouseout`[2] on the iframes beneath. We note that to click a button or write in a text area, the user does not require access to all the pixels for that object. If the user can click on any part of the button accessible to him, he is able to initiate the event on that button. Similarly, if the user can access any part of the text area, he can type in it even though part of the text area is hidden behind an opaque element. As described in Section 3.1, a user attacker can break a website's behavior and access information by exploiting this security flaw in the Android and Opera Mini browsers.

The iPhone Safari browser also allows a user to access restricted content by exploiting its non-compliance to the proposed policy. We observe the exact same vulnerability in iPhone Safari as its desktop counterpart. If two or more iframes of any origin overlap, the user is able to scroll all the contiguous transparent iframes from the top.

We observe that incorrect handling of user access in overlapping elements is not restricted only to transparent display elements. If two opaque images overlap, the bottom image is accessible through the top image in the overlapped area on the Android and Opera Mini browsers. We note that the bottom image is accessible only when the corners of the images overlap. Events such as `onclick` and `onmouseover` are executed through the opaque overlapped area of the images. This problem may subject a user to clickjacking while simply browsing a webpage containing cross-origin overlapped images.

The Opera Mini and Nokia Mini-Map browsers present more vulnerabilities in handling user access to overlapping opaque elements. Even if the opaque image on top has an `onlick` event defined, the button below the image responds in the Opera Mini browser completely ignoring the click event of the top element. Similarly, in the Nokia Mini-Map browser, the `onclick` events of the text areas and links below an opaque image get precedence while completely ignoring the `onlick` event of the top image. If the top opaque image does not have an event defined, elements such as buttons and text areas situated below the image are accessible to the user in both browsers. As explained in Section 3.1, this vulnerability makes the Opera Mini and the Nokia Mini-Map browsers susceptible to the login CSRF and snooping attacks. The Firefox Mobile and Android browsers have a similar vulnerability. They allow access to hidden text area through opaque images. We note that the login CSRF attack is possible on the Android and Firefox Mobile browsers even though they do not allow user access to a button below an opaque image. We successfully built the login CSRF attack described in Figure 1 on the Android browser. The attack is possible because pressing the 'enter' key on the keyboard can also log a user in the attacker's account once the user has filled in the username and password. Therefore, instead of asking the user to press the fake 'verify' button on the screen, an attacker can simply ask the user to 'Press Enter' on their keypad after solving the CAPTCHAs. Therefore, the Opera Mini, Nokia Mini-Map, Android and Firefox Mobile browsers are susceptible to the both the login CSRF and the snooping attacks. We note that the same attack would work on all the aforementioned vulnerable browsers.

The Blackberry browsers built on Mango and Webkit engine correctly follow the user access policy for overlapping elements for our test cases. Similar to the desktop browsers, only the top website principal holds the ownership of user access to the overlapped pixels in Blackberry browsers.

The IE 8 browser on Windows 7 desktop does not support same or cross-original overlapping transparent iframes. The top element of an overlay is made opaque even though its transparency property is set to true. Therefore, when two or more transparent iframes overlap, both the display and user

[2]The `onmouseover` and `onmouseout` events are supported on the majority of mobile browsers.

	Landlord							
	Google Android	iPhone Safari	Opera Mini	Nokia Mini-Map	Firefox Mobile	Windows IE	Blackberry Mango	Blackberry Webkit
position (x,y,z)	RW	RW	RW	RW	RW	RW	RW	RW
dimensions (height, width)	RW	RW	RW	RW	RW	RW	RW	RW
pixels	R^*	–	R^*	R^*	R^*	–	–	–
URL location	W	W	W	–	W	W	W	W

Table 4: Policies for a landlord to access a cross-origin tenant's position, dimensions, pixels and URL location. R^*: The Android, Opera Mini and Firefox Mobile browsers allow a landlord to read the user interaction with its cross-original tenant. This vulnerability breaches the access control policy for the tenant's 'pixels' allowing the landlord to launch a snooping attack.

access of the overlapped area is owned by the top element. Similar behavior is seen in the IE browser on Windows 7 phone OS. Although both browsers follow a different display ownership policy for transparent iframes, they both follow our proposed principal for transparent and opaque elements.

3.3 Discussion

Our experiments show that the majority of the popular mobile browsers do not handle the user access to overlapping elements correctly. *Five* out of the *eight* candidate mobile browsers are vulnerable to a range of attacks such as login CSRF, clickjacking and elevation of user privileges due to not following the user access policy correctly. We note that these vulnerabilities are prevalent in the mobile browsers whereas desktop browsers (excluding Safari) implement the policy correctly. This result defies the expected assumption that browsers from the same vendors automatically implement similar security policies. Although Safari desktop and iPhone Safari show the same vulnerability, the Opera Mini and Firefox Mobile browsers behave very differently when compared to their desktop counterparts.

If a landlord can snoop on the interaction between a user and the landlord's cross-origin tenant (Section 3.1.3), we consider that as a violation of the access control policy due to the ability of the landlord to read the private content of the tenant. Table 4 shows our evaluation of the access policies for a landlord on different browsers. We note that allowing READ access to a tenant's pixels (user interaction) makes the Android, Opera Mini, Firefox Mobile, Nokia Mini-Map and iPhone Safari browsers susceptible to snooping.

We note that following our proposed policy does not prevent all possible attacks when cross-origin elements overlap. For example, if a malicious transparent iframe is placed exactly on top of the login box of an honest landlord, according to our policy, the transparent iframe will own the control of user access to the overlapped area. When the user enters his login credentials in the login box, he actually submits his credentials to the malicious iframe. Such attacks can be prevented if our policy is extended to include the 'opaque overlay policy' proposed by Wang et al [36]. The opaque policy allows only same origin windows to transparently overlay with one another. Additionally, cross-origin windows are allowed to overlay only when the top window is opaque. Therefore, attacks caused due to overlapping of transparent cross-origin elements can be prevented using the opaque policy. However, the opaque overlay policy does not allow cross-origin drop down menus or cross-origin transparent overlapping advertisements. This behavior restricts browser functionality and can break website logic. Accordingly, most modern browsers do not follow the opaque overlay policy; however, the majority of browsers do support overlay of cross-origin transparent elements. We also note that the opaque overlay policy does not completely provide cross-principal event protection in overlapped elements by itself. If a cross-principal opaque element at the top shares its control of user access with the bottom element, the logic of the webpage is broken and it becomes vulnerable to attacks. We therefore conclude that defining policies only for display of overlapped elements is not sufficient. Uniquely identifying the owner of the user access of overlapped pixels in addition to ownership of display is crucial for cross-principal event protection in overlapped elements.

We note that even if our proposed policy is extended to include the opaque policy, imitation attacks caused by overlapping of opaque elements are not solved. For example, consider a malicious iframe imitating the Google login page overlapping on top of the actual Google login iframe. The opaque policy will allow this cross-origin overlap and our policy would restrict the control of user access to the malicious iframe. A user will perceive the malicious iframe as the real Google login page and will be easily subjected to a password stealing attack. A potential solution to this problem could be to change the browser GUI such that the user sees the actual URI location of the content he interacts with all the time. This solution will require changes to all the existing browsers. Another potential solution to prevent all the security issues with overlapped elements is to remove the support for overlapping elements in browsers. This is an extreme solution and would break the logic of a large number of websites. For example, 6.2% websites out of the top 100,000 contain at least one pair of overlapping frames [31] and would be broken if overlapping elements are not supported by browsers. Additionally, there may be other websites containing overlapping display elements other than just iframes. Therefore, we deem both the plausible solutions to imitation attacks to be difficult to realize.

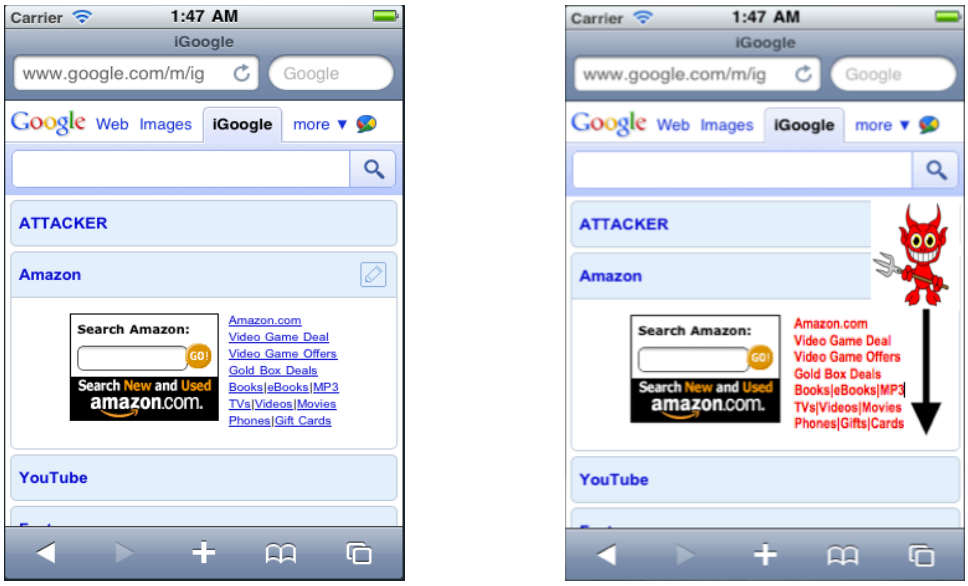

Figure 3: **Left image**: Layout of the malicious and honest widgets on the mashup webpage. 'ATTACKER' is a malicious widget and Amazon and YouTube are honest widgets; **Right image**: The browser allows a cross-origin tenant to read and write its own dimensions. The malicious widget expands its own 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 and Opera Mini browsers.

4 Display Access for Cross-origin Elements

In Section 3, we concluded that to completely secure browsers from attacks caused by overlapping elements, the extreme solution would be to remove the support for overlapping elements from browsers. In our evaluation of display in mobile browsers, we observed that mobile browsers bring in new security challenges even when there are no overlapping elements. In this section, we study vulnerabilities in mobile browsers when each display pixel's visual and user access properties are completely owned by only one principal.

Complex websites often contain one or more cross-origin tenants in the form of advertisements or gadgets. In such a setting, both the landlord and the tenant have to protect themselves from each other. Websites rely on browsers to provide access control guarantees when cross-origin principals interact. Wang et al [36] defined an access control matrix for display in a cross-origin landlord-tenant setting for the Gazelle browser as shown in Table 2. Previous studies [31] have evaluated compliance with the display access control policies proposed by Gazelle on desktop browsers. We advocate the display access control policies proposed in Gazelle in a cross-origin landlord-tenant setting and evaluate the compliance with the policy on mobile browsers.

Similar to earlier work on access control policy for display [36, 31], we evaluate the read and write access control policy for position, dimensions, pixels and URL location of a cross-origin tenant. According to the access control matrix defined in Gazelle, when a landlord rents an area on his webpage to a cross-origin tenant, he should be able to control the position and dimensions of the tenant. However, once rented, the landlord should not be able to read the private content of the tenant's pixels, the URL location of the tenant or change the tenant's logic by writing over its pixels. The tenant should not be able to resize or position itself on the landlord's screen.

If the access control policies in a cross-origin landlord-tenant setting are not followed, honest principals become vulnerable to the following attacks.

4.1 Attacks

If a browser incorrectly implements the access control policy for dimensions of a tenant, a malicious tenant is able to attack honest landlord and the user. We present two potential attacks on this browser vulnerability.

4.1.1 Phishing

A browser allowing a cross-origin tenant to read and write its own dimensions is vulnerable to phishing attacks.

Consider the iGoogle mashup as shown in Figure 3. A mashup combines data or functionalities from more than one source to provide a new service. The landlord in a mashup is the principal hosting all the functionalities in the form of widgets, which are generally tenants from other origins. Any user having an account with the landlord is allowed to upload and publish a new widget on the mashup website. Other users can then add the widget to their own profiles on the mashup website.

	Tenant							
	Google Android	iPhone Safari	Opera Mini	Nokia Mini-Map	Firefox Mobile	Windows IE	Blackberry Mango	Blackberry Webkit
position (x,y,z)	–	–	–	–	–	–	–	–
dimensions (height, width)	RW^*	RW^*	RW^*	R	R	–	R	R
pixels	RW	RW	RW	RW	RW	RW	RW	RW
URL location	RW	RW	RW	RW	RW	RW	RW	RW

Table 5: Policies for a tenant of a cross-origin landlord to access its own position on the landlord's page, its dimensions, its URL location and its pixel content. RW^*: Android, iPhone Safari and Opera Mini browsers allow a cross-origin tenant to write its own dimensions and thus become susceptible to phishing attacks.

Each widget in the mashup is contained inside an iframe on the landlord's page. Consider a malicious widget *ATTACKER*. After an honest user adds the widget to his profile, it is placed "North" of an honest widget Amazon as shown in Figure 3. The honest user browses online deals through the Amazon website and makes purchases of items of his choice. The intention of the malicious tenant is to navigate an honest user to a website of his choice. Being aware of the lack of enforcement of the display policy by the user's browser, the malicious tenant alters his dimensions as shown in Figure 3. The attacker 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 on the mashup website, 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 widgets present on the victim's personal profile and the layout of the profile. Mashup websites generally include some default widgets on a new profile. The attacker can masquerade as any of the default widgets on the 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.

4.1.2 Password Stealing

Consider a malicious advertisement situated to the 'North' of the login box of an honest website. The malicious advertisement 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 down on the user's phone screen. Since 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.

4.2 Experimental Evaluation

We built test cases containing a cross-origin tenant on a landlord. We verified the compliance of the candidate browsers with access control policies for four attributes of the tenant window: position, dimensions, pixels and URL location.

Table 5 lists the access policies for cross-origin landlord and tenant in the eight mobile browsers. *Three* out of the *eight* mobile browsers do not correctly follow the access control policy for a tenant's dimensions when cross-origin tenant and landlord interact. The Android, Opera Mini and iPhone Safari browsers allow a tenant of a different origin than that of the landlord to write its own dimensions. All three 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 its own 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. However, being able to push legitimate content out of the user's view allows a malicious tenant to launch phishing and password stealing attacks. We built the phishing attack described in Section 4.1.1 on iGoogle and launched it successfully on the Android, Opera Mini and iPhone Safari browsers.

The Internet explorer browser on Windows 7 phone OS, Blackberry browser built on Mango engine and the Blackberry browser built on Webkit comply with the policies for cross-principal display access control correctly.

4.3 Discussion

Our experiments show that attacks such as phishing and password stealing are possible against *three* (Android, Opera Mini and iPhone Safari) out of *eight* mobile browsers. A tenant attacker can launch a phishing attack on any website supporting uploading of user generated widgets in iframes by expanding his iframe. An iframe holding more content than can be displayed is not perceived as a malicious iframe. Browsers are expected to provide scrolling and restrict the write access to the tenant's dimensions to the landlord. Scrolling allows a user to access all the content of the iframe and also ensures that the tenant resides only in the area rented out by the landlord. Similar to desktop browsers, mobile browsers do not treat iframes holding more content than can be displayed as malicious. Therefore, the phishing attack is fairly easy to launch on any mashup website allowing user generated content. More importantly, this demon-

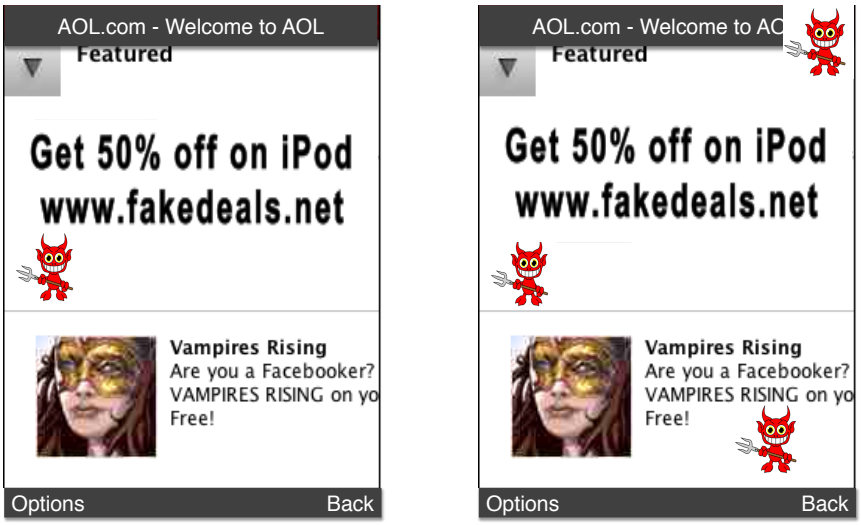

Figure 4: **Left image**: *www.aol.com* web page containing a cross-origin malicious advertisement. The browser displays only the 'title' of the page and does not display the address bar.; **Right image**: Due to the top-level frame navigation policy, the malicious advertisement can redirect the top-level 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 browser is the most susceptible to this attack. However, all other mobile browsers are also susceptible to this attack due to address bar not being persistently available while browsing.

strates a *significant tension* between usability and display security in mobile browsers.

The same vulnerabilities exist across mobile browsers built on different rendering engines. For example, the write access to self dimensions for a cross-original tenant is a vulnerability seen on the Opera Mini browser built using Presto engine and the Android and iPhone Safari browsers built using Webkit. None of the desktop browsers (excluding Safari desktop) in our tests show the vulnerabilities found in mobile browsers. Therefore, we conclude that the non-compliance to display access control policies is prevalent in popular mobile browsers. This observation further strengthens the argument that mobile browsers introduce new security problems and are not simply mini versions of the desktop software.

5 Top-Level Frame Navigation

Navigation has been considered as part of display in previous studies [31]. Navigation is defined as an action to change the location of a window and point it to a new website. When cross-origin principals interact, a principal's ability to navigate principals of other origins is governed by the navigation policies specified by the browser.

Barth et al. have previously noted the top-level frame navigation policy. They state that most browsers allow top-level frames to be navigated by anyone, because they have an address bar [10]. This implies that it is crucial for a user to *always* view the address bar in order to make the decision of credibility of the website and identify a potential phishing attack. All desktop browsers allow a user to *always* view the top-level window's address bar. Therefore, the top-level frame navigation policy can possibly prevent phishing attacks on a careful user on desktop browsers.

We argue that mobile browsers are considerably different in handling address bar of the top-level window. Due to the smaller screen size, once the page is rendered, the address bar is minimized and cannot be viewed by the user in most mobile browsers. If the address bar of the top-level window is not *always* visible, the following attack is possible.

5.1 Attacks

A tenant attacker (descendant) can launch a phishing attack very easily 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 web page *www.honest.com* consisting of a malicious cross-origin advertisement as shown in Figure 4. The `onload` event of the advertisement is to navigate the top-level window to *www.attacker.com*, which looks exactly the same as *www.honest.com* and contains malicious content. When the advertisement 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.

5.2 Experimental Evaluation

We confirmed our initial observation by evaluating the mobile and desktop browsers. All *thirteen* mobile and desktop browsers allow a descendant principal of any origin to navigate the top-level window to any source. *Five* desktop browsers *always* display the address bar in the window.

The iPhone Safari browser minimizes the top-level address bar for better usability once a page is rendered. The address bar disappears from view once the user starts interacting with content on the page. This behavior is seen in all the mobile browsers except Internet Explorer Mobile 8 and Nokia Mini-Map. Internet Explorer Mobile 8 always displays the address bar, even 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 web page in the landscape mode.

In the Nokia Mini-Map browser, the address bar of the top-level window is *never* accessible to the user while browsing. We tried all the functionalities provided in options of the web browser to determine the URL location of the currently rendered page. However, the Nokia Mini-Map browser does not provide a functionality to access the address bar. This makes the Nokia browser the most susceptible to phishing attacks by navigation of top-level window to malicious pages, since the user can never detect the attack.

5.3 Discussion

Zooming and scrolling are two very important functions of mobile browsers to perform useful actions. Since these functionalities push the address bar of a web page 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. A more restrictive policy, such as Gazelle's top-level frame navigation policy [11] allows only the top-level window's tenant and the user to navigate the top-level window. This approach would better balance issues of usability, specifically screen real-estate, and security.

6 Other Experiments and Results

We evaluate mobile browsers for *two* other previously studied navigation policies for desktop browsers.

Earlier studies have advocated the descendant [21] and child policies [36] for frame navigation. The descendant policy allows navigation of any descendant of a window. The child navigation policy is more restrictive than the descendant navigation policy as the child policy allows only the direct children of a window to be navigated. However, the descendant navigation policy has been shown to be at conflict with the DOM's Same Origin Policy (SOP) [36, 31]. The descendant policy allows a malicious landlord to change the logic of a cross-original tenant by navigating the grandchildren frames inside the tenant to a new source. This behavior violates the policy for a landlord's access to the tenant's private information since the grandchildren frames of the landlord are private to the tenant. *Seven* mobile and all *five* desktop browsers follow the descendant policy for frame navigation. The Blackberry browser with the Mango rendering engine is the only browser, which implements the child navigation policy. As a result, we conclude that all but one of the mobile browsers also suffer from similar vulnerabilities as their desktop counterparts.

We also studied the history property associated with the window object of each browser. All browsers except Opera Mini, Windows Mobile IE and Blackberry Mango allow a descendant of any origin to navigate the top-level window back and forth in its array of history URLs. A malicious descendant can exploit this vulnerability. For example, a user can be tricked into making multiple purchases of the same product [31]. *Eight* out of the *thirteen* desktop and mobile browsers used for our studies are vulnerable to this problem. We note that the desktop browsers and their corresponding mobile versions show the same properties for navigation. For example, both the desktop and mobile versions of Opera and Windows IE browsers restrict cross-origin access to top-level window's history. The desktop and mobile versions of the Safari and Firefox browsers allow any cross-origin descendant principal to navigate the top-level window in its history.

7 Observations

In this work, we evaluated display security policies in mobile browsers when display elements overlap, access each other's resources and are navigated to a new location. We observed that iframes are a major source of display-related security issues in mobile browsers. Iframes are used extensively on the Web, with more than 40% of the top 100,000 websites embedding at least one iframe [31]. These websites are primarily meant for desktop browsers and all security considerations are made based on the desktop environment. However, all these webpages may become susceptible to a range of attacks when rendered on mobile browsers. For example, 40% webpages containing iframes would be susceptible to attacks due to the iframe's tenant being able to write its own dimensions in the Android, Opera Mini and iPhone Safari browsers. Moreover, 1.2% of the top 100,000 sites with transparent, overlapping cross-origin iframes [31] would potentially suffer from overlapping vulnerabilities presented in Section 3.1. This gives us an initial estimate of the backward compatibility cost of fixing the security issues due to overlapping iframes if browser vendors decide to use more stringent overlap policies such as opaque policy [36]. We plan to improve such estimates to include mobile websites as part of our future work.

Some interesting observations can be made from the comparison between desktop and mobile versions of the same browsers. While it is expected that browsers from the same vendor (such as Safari, Firefox, Opera and IE in our evaluation) would have similar security policies for both their desktop and mobile versions, we found that browsers devi-

ate considerably from this expectation. With the exception of the mobile and desktop version of the IE browser, all the other desktop and mobile browser pairs from the same vendor do not implement the same security policies for display. For example, the Opera Mini browser does not follow the proposed user access policy and also allows a cross-origin tenant to write its own dimensions allowing a wide range of attacks. The Opera desktop browser is not susceptible to any of the attacks shown on Opera Mini due to correctly complying with display policies. The mobile and desktop versions of Firefox and Safari browsers also show differences in adhering to display security policies.

Another interesting observation is that the display-related security issues on mobile browsers are not restricted to one rendering engine. For example, the vulnerabilities caused due to incorrect handling of overlapping elements are present on browsers built using the Presto, Gecko and Webkit engine. This implies that similar security design issues exist across browser engines. Moreover, we observed the popular browsers to be more vulnerable to attacks as compared to others. For example, the Opera Mini browser is the most popular browser on mobile phones and we discovered Opera susceptible to most of the potential attacks.

Smaller screen size and optimized functionalities for constrained hardware as compared to desktop browsers is the possible reason behind the rendering problems found in this paper. However, we believe that rendering is only one of the many aspects that differentiate mobile browsers from their desktop counterparts. There are other differences in desktop and mobile environments such as lesser processing power and lesser memory, which may also play an important role in defining a user's web experience on a mobile phone. For example, since flash consumes considerable memory and processing power, it was not available on mobile phones for a fairly long time. With mobile phones becoming all purpose computing systems, the gap between desktops and mobile phones as two computing systems is closing very fast. However, certain differences such as screen size will remain the same to continue supporting portability. We plan to extend our work to analyze other differences between the two environments, including session management and handling of cookies; and how these differences influence the effectiveness of their security policies.

Despite our effort to be comprehensive, we may have missed some display-related issues existing in today's browsers. We hope our work to be a start for a community effort on mapping out the full set of safe browser access control policies for mobile platforms.

8 Related Work

Desktop browsers have been shown to be vulnerable to a variety of attacks in the past including Cross Site Scripting [18], Cross Site Request Forgery [20], clickjacking [4, 5, 29] and phishing. In addition to weak security policies, implementation errors in the browser code [22], inconsistencies in access control policies [31], absence of wide deployment of policies [37] and incorrect handling of privileges in browser extensions [19] further increase the threats to the browser and the user. To protect browsers from attacks, a range of defenses have been proposed including implementing new HTTP headers [20], enforcing new security policies [25, 32, 27] and algorithm [18, 16] and development of tools to find potential security vulnerabilities in browsers [17].

The increasing interplay of cross-domain principals by complex websites demands resource handling in the browser similar to an operating system. The OP Web browser [26] was the first to design a small browser kernel to enforce new browser security features and handle resources. Gazelle [36] and Chrome [11] also proposed new browser architectures for separating the functionality of the browser from security mechanisms and policies. King et al [33] continued the design philosophy through the Illinois browser OS by directly mapping browser abstractions to hardware abstractions.

Smart phones have become multipurpose computing platforms. Portability makes them an attractive platform to host applications and browse Web. However, malicious mobile applications can affect user privacy [24, 23] and potentially harm the cellular network [35]. Increasing user base has made mobile browsers an attractive target for attackers [30, 9, 6, 13, 3, 7]. Researchers have already begun to think about defending against attacks on mobile phones using smart CDNs [28]. 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 browsers has not yet been performed.

9 Conclusion

Mobile web browsers have been long thought of as the mini-versions of their desktop counterparts. In this paper, we take the first step towards analyzing the differences between mobile and desktop browsers from the point of view of security. We focus our analysis on display-related security issues. We identify two separate classes of security problems in mobile browsers and devise several real-world attacks against them. Additionally, we identify an existing security policy for display on desktop browsers that is inappropriate on mobile browsers. We perform a comprehensive analysis of *eight* mobile and *five* desktop browsers to determine the display-related differences between the two. We observe that desktop browsers are more compliant with display-related security policies as compared to their mobile counterparts. Moreover, we show that limitations (especially screen size) make the failure to enforce such policies even more dangerous in the mobile space. We conclude that not only do mobile browsers bring in new security challenges but also that they are not mini-versions of their desktop counterparts.

References

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

[2] Android and iPhone browser wars, Part 1: WebKit to the rescue. http://www.ibm.com/developerworks/opensource/library/os-androidiphone1/.

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

[4] Chrome, Firefox get clickjacked. http://www.zdnet.com.au/chrome-firefox-get-clickjacked-339294633.htm/.

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

[6] iPhone overflow clickjacking. http://ejohn.org/blog/clickjacking-iphone-attack/.

[7] iPhones Safari Vulnerable To DoS Attacks. http://www.iphonebuzz.com/iphone-safari-dos-bug-discovered-162212.php.

[8] Mobile Browser Market Share. http://gs.statcounter.com/#mobile_browser-ww-monthly-201001-201101.

[9] Overflow clickjacking. http://research.zscaler.com/2008/11/clickjacking-iphone-style.html.

[10] Protecting Browsers from Frame Hijacking Attacks. http://seclab.stanford.edu/websec/frames/navigation/.

[11] The security architecture of the chromium browser. http://blogs.blackberry.com/2010/08/blackberry-torch/.

[12] W3C - CSS Mobile Profile 2.0. http://www.w3.org/TR/2008/CR-css-mobile-20081210/.

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

[14] 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.

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

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

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

[18] A. Barth, J. Caballero, and D. Song. 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.

[19] A. Barth, A. P. Felt, P. Saxena, and A. Boodman. Protecting Browsers from Extension Vulnerabilities. In *Proceedings of the 17th Network and Distributed System Security Symposium (NDSS)*, 2010.

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

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

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

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

[24] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. 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.

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

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

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

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

[29] G. Rydstedt, E. Bursztein, D. Boneh, and C. Jackson. 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.

[30] G. Rydstedt, B. Gourdin, E. Bursztein, and D. Boneh. 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.

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

[32] S. Tang, C. Grier, O. Aciicmez, and S. T. King. Alhambra: a system for creating, enforcing, and testing browser security policies. In *Proceedings of the International Conference on World Wide Web (www)*, 2010.

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

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

[35] P. Traynor, M. Lin, M. Ongtang, V. Rao, T. Jaeger, T. La Porta, and P. McDaniel. 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.

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

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