Computer Standards & Interfaces 49 (2017) 71–78

Contents lists available at ScienceDirect

Computer Standards & Interfaces

journal homepage: www.elsevier.com/locate/csi

Mobile cloud security: An adversary model for lightweight browser security

Shasi Pokharel [a], Kim-Kwang Raymond Choo [b,a,*], Jixue Liu [a]

[a] School of Information Technology & Mathematical Sciences, University of South Australia, GPO Box 2471, Adelaide, SA 5001, Australia
[b] Department of Information Systems and Cyber Security, University of Texas at San Antonio, One UTSA Circle, San Antonio, TX 78249-0631, USA

ARTICLE INFO

Article history:
Received 7 March 2016
Received in revised form
18 August 2016
Accepted 8 September 2016
Available online 9 September 2016

Keywords:
Mobile cloud security
Lightweight browser security
UC Browser
Dolphin
CM Browser
Samsung Stock Browser

ABSTRACT

Lightweight browsers on mobile devices are increasingly been used to access cloud services and upload / view data stored on the cloud, due to their faster resource loading capabilities. These browsers use client side efficiency measures such as larger cache storage and fewer plugins. However, the impact on data security of such measures is an understudied area. In this paper, we propose an adversary model to examine the security of lightweight browsers. Using the adversary model, we reveal previously unpublished vulnerabilities in four popular light browsers, namely: UC Browser, Dolphin, CM Browser, and Samsung Stock Browser, which allows an attacker to obtain unauthorized access to the user's private data. The latter include browser history, email content, and bank account details. For example, we also demonstrate that it is possible to replace the images of the cache in one of the browsers, which can be used to facilitate phishing and other fraudulent activities. By identifying the design flaw in these browsers (i.e. improper file storage), we hope that future browser designers can avoid similar errors.

1. Introduction

In recent years, we have seen a rapid shift in Internet browsing behaviors from the use of personal computers (PCs) to mobile devices, particularly accessing cloud services and storing data in the cloud [20,45,23]. In other words, Internet browsing is increasingly being conducted on mobile devices [51]. This has also resulted in an increasing use of lightweight browsers on mobile devices.

Lightweight browsers are popular for their speedy resource loading capabilities, particularly for viewing large media files or for gaming. However, the trade-off is reduced user functionalities and weakened security mechanisms [57,58]. For example, basic browser security requirements defined by W3C [46] implemented in typical browsers, such as Google Chrome and Mozilla Firefox, may not be installed on the lightweight browsers [3].

Browsers are security sensitive applications, as they are able to access personally identifiable information (PII) and sensitive data such as bank account details. Browser communications can be targeted at various stages of the communication, such as on client devices, during network transmission, and at the server. Security issues and mitigation strategies relating to the network and the server have gained significant interest (see [14,15,52]). The security of browsers in mobile devices, however, appears to be an understudied area. For example, the question whether cache and other files are securely stored by browsers so that they cannot be accessed by unintended person or apps has not been well studied (e.g. are cache and other files encrypted or stored with the appropriate file permission?).

In this paper, we attempt to evaluate the security of user information stored by the lightweight browsers on mobile devices. Using an adversary model adapted from the security literature, we examine four popular lightweight browsers for Android device and reveal previously unpublished vulnerabilities. We regard the contributions to be two-fold:

1) An adversary model designed to study the security of lightweight mobile browser; and
2) Identification of previously unpublished vulnerabilities in four lightweight browsers.

The rest of the paper is organized as follows. Background materials and related literature are described in Sections 2 and 3, respectively. In Section 4, we present the proposed adversary model and the prototype app. The experiment setup and findings are respectively outlined in Sections 5 and 6. The last section discusses potential mitigation strategies and concludes the paper.

* Corresponding author at: Department of Information Systems and Cyber Security, University of Texas at San Antonio, One UTSA Circle, San Antonio, TX 78249-0631, USA. Tel.: +1.210.458.7876.
E-mail address: raymond.choo@fulbrightmail.org (K.-K. Choo).

http://dx.doi.org/10.1016/j.csi.2016.09.002
0920-5489/

2. Background: Mobile browsers

Browsing a webpage requires the loading of multiple sets of resources, such as HTML, CSS, JavaScript and media files. For example, according to Wang et al. [54], loading of such resources can be slower on mobile devices than on PCs due to the architectural differences and computational constraints.

Speed during website browsing is a key user concern. For example, a one second delay in webpage loading could reportedly result in 11% reduction in webpage views and 16% reduction in customer satisfaction [39]. Similar observations were echoed in the studies by Amazon [33] and Google [10].

Lightweight browsers apply client side efficiency solutions to improve the browsing speed, and consequently user's quality of experience. This includes creating a larger cache storage and avoiding any plugins that can delay the loading of web resources. Cache is the temporary storage to save downloaded web resources. If a user attempts to access a previously accessed same page or URL, the browser checks whether the content exists in the cache. If the contents exist, then the browser loads the resources from the cache; thus, saving time and network resources.

Popular browsers, such as Google Chrome, Mozilla Firefox, and Opera, use standard Web Storage to store cache data. Web Storage was introduced as a part of HTML5 and is being standardized by World Wide Web Consortium (W3C). Web Storage contains two major parts, namely: Local Storage and Session Storage, whose behavior is similar to that of persistent cookies and session cookies, respectively. Session storage stores web resources until the webpage is open. In the case of Local Storage, the generated cache remains on the device even when the browser is closed [55].

In both PC and mobile device environments, Web Storage is considered more secure than the native browser cache. According to W3C, Web Storage can be used to store sensitive user information, if implemented properly [55]. On Android devices, Web Storage typically uses the device's internal storage (e.g. /data/data/PackageName/ directory). Therefore, the items stored in these cache storage cannot be accessed by other users or apps, with the exception of the owner's app.

However, Web Storage is limited by cache size. For general use, W3C recommends the use of 5MB storage size per website, but this can be reduced when implemented on mobile devices. Lightweight browsers mostly rely on large cache storage to improve the browser's loading speed. Therefore, these browsers store large amount of cache data outside of Web Storage, often in external storage (e.g. SD card).

For Android devices, internal storage is generally considered a more secure storage location for application data, because, by default, stored data can be accessed or modified only by the creator app. In comparison, any resources, stored in external storage can be accessed, modified or deleted by any applications that have READ_ EXTERNAL_STORAGE Permission [16].

3. Related work

Web (application) security has been a research focus for a number of years [40]. Browsers for PCs, laptops and mobile devices share the underlying rules for loading webpages and communicating with servers. Therefore, existing literature on browser security tend to be focused on 'traditional' browsers (for PCs and laptops), as well as focusing on either network security or on detecting malicious websites (see [18,19,21,24,50]).

In 2014, Wadkar, Mishra and Dixit proposed the 'system call' monitoring approach to prevent information leakage from the browser [53]. System call is an interface between the browser application and Operating System (Linux) kernel, which is invoked during the execution of browser process. The researchers proposed an intermediate layer between the Kernel and the application layer that controls the system calls and filters the personal information being leaked during the browsing.

Virvilis et al. [52] evaluated the effectiveness of the Blacklist filtering approach on browsers, designed to prevent users from visiting rouge or malicious webpages. In another related study, Amrutkar et al. [2] presented a threat model, which allows the discovery of architectural weakness on mobile devices and browsers. The researchers demonstrated that attack vectors, such as display ballooning, Cross Site Request Forgery (CSRF) and clickjacking, can be used for phishing or directly stealing information from the users' device (Table 2).

More recently in 2015, Amrutkar, Traynor and van Oorschot [3] evaluated the security indicators (based on the security guidelines of W3C – [46]) used in popular mobile browsers. For example, they check to determine whether the browser displays identity of the site owner and certificate issuer and whether the browser uses the

Table 1
Lightweight browsers.

S. no.	Browser Name	Version No.	in Google Play Store downloads (in millions; as of Sep 2015)	Remarks
1.	UC Browser	10.6.2	100–500	
2.	Dolphin	11.4.19	50–100	
3.	CM Browser	5.20.06	10–50	
4.	Samsung Stock Browser	N/A	N/A	Pre-installed with Samsung mobiles, so total user number and application version cannot be identified.

Table 2
Targeted cache and file storage locations of the browsers in the study.

Browser	Targeted Cache Location	Important Contents
Dolphin	/sdcard/TunnyBrowser/cache/speeddial_covers	URLS saved as speed dial
	/sdcard/TunnyBrowser/cache/tablist_cache	screenshot image files
	/sdcard/TunnyBrowser/cache/webViewCache	All cache files (HTML, CSS, JavaScript, media)
UC Browser	/sdcard/UCDownloads/cache/	All cache files
	/sdcard/UCDownloads/config/	TrafficStatus.db; contains client server communication timing and response
	/sdcard/UCDownloads/offline/	ApplicationCache.db; contains data for cache loading management
CM Browser	/sdcard/CheetahBrowser/.data/	Browsers URL history
Samsung Stock Browser	/data/data/com.sec.android.app.sbrowser/files/	Screenshot image files