On the Usability (In)Security of In-App Browsing Interfaces in Mobile Apps

Zicheng Zhang
Singapore Management University
Singapore
zczhang.2020@phdcs.smu.edu.sg

Daoyuan Wu*
The Chinese University of Hong Kong
Hong Kong, China
dywu@ie.cuhk.edu.hk

Lixiang Li†
miHoYo Co., Ltd.
China
lixiang.li@mihoyo.com

Debin Gao
Singapore Management University
Singapore
dbgao@smu.edu.sg

ABSTRACT

Due to the frequent encountering of web URLs in various application scenarios (e.g., chatting and email reading), many mobile apps build their in-app browsing interfaces (IABIs) to provide a seamless user experience. Although this achieves user-friendliness by avoiding the constant switching between the subject app and the system built-in browser apps, we find that IABIs, if not well designed or customized, could result in usability security risks.

In this paper, we conduct the first empirical study on the usability (in)security of in-app browsing interfaces in both Android and iOS apps. Specifically, we collect a dataset of 25 high-profile mobile apps from five common application categories that contain IABIs, including *Facebook* and *Gmail*, and perform a systematic analysis (not end-user study though) that comprises eight carefully designed security tests and covers the entire course of opening, displaying, and navigating an in-app web page. During this process, we obtain three major security findings: (1) about 30% of the tested apps fail to provide enough URL information for users to make informed decisions on opening an URL; (2) nearly all custom IABIs have various problems in providing sufficient indicators to faithfully display an in-app page to users, whereas ten IABIs that are based on Chrome Custom Tabs and SFSafariViewController are generally secure; and (3) only a few IABIs give warnings to remind users of the risk of inputting passwords during navigating a (potentially phishing) login page.

Most developers had acknowledged our findings but their willingness and readiness to fix usability issues are rather low compared to fixing technical vulnerabilities, which is a puzzle in usability security research. Nevertheless, to help mitigate risky IABIs and guide future designs, we propose a set of secure IABI design principles.

*Daoyuan Wu is the corresponding author.
†Lixiang Li was a MSc student when he conducted this study through the Advanced Research and Development Project course at The Chinese University of Hong Kong.

RAID '21, October 6–8, 2021, San Sebastian, Spain

ACM ISBN 978-1-4503-9058-3/21/10…$15.00
https://doi.org/10.1145/3471621.3471625

CCS CONCEPTS

• **Security and privacy → Mobile platform security**.

KEYWORDS

Android Security; Usability Security; WebView Security

ACM Reference Format:
Zicheng Zhang, Daoyuan Wu, Lixiang Li, and Debin Gao. 2021. On the Usability (In)Security of In-App Browsing Interfaces in Mobile Apps. In *24th International Symposium on Research in Attacks, Intrusions and Defenses (RAID '21), October 6–8, 2021, San Sebastian, Spain.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3471621.3471625

1 INTRODUCTION

Nowadays, mobile applications (or apps) are heavily used in our daily life. Although most app functionalities are self-contained, it is not uncommon for users to open (external) web URLs in their app UIs (user interfaces). For example, a user may need to open a URL sent from her friends in a chat app like *Whatsapp* or *WeChat*, or need to open a URL embedded in an email when using *Gmail*. To satisfy such URL opening requirements in non-browser apps, one could offload the task to the system built-in browser apps. While this is simple for developers, it hurts user-friendliness due to the potential constant switching from the subject app to a system browser app. As a result, many high-profile apps choose to provide their own *in-app browsing interfaces*, or IABIs, for a seamless user experience.

However, if not designed or customized well, IABIs could introduce serious usability security issues. The major reason is that IABIs are typically simplified implementations of browsing interfaces lacking security indicators as opposed to "full-service" and standalone browsers with well-thought usability security designs. For example, an IABI may not display the full URL domain name or simply miss the HTTP(S) indicator. Motivated by this intuition, we conduct the first empirical study in this paper on the usability (in)security of in-app browsing interfaces in both Android and iOS apps. To this end, we collect and analyze a dataset of 25 high-profile mobile apps that contain IABIs, such as *WeChat, Twitter, Gmail, LinkedIn*, and *Reddit*. To make our results representative, these apps are selected from five common app categories, including Chat, Social, Mail, Business, and News.

Atop this dataset, we perform a systematic analysis that comprises eight carefully designed security tests (T1~T8 in three categories) and covers the entire course of interacting with an in-app web page in IABIs including page opening, displaying, and navigating. First, before a user opens a URL, we test whether the subject app provides sufficient URL information to enable end users to make informed decisions on opening the URL in a trustworthy manner (T1). Second, after the web page is loaded, we test whether the IABI provides enough security indicators for end users to validate the trustworthiness of the displayed page. This includes whether the URL itself (T2), an HTTPS (secure) indicator (T3), and an HTTP (insecure) warning (T4) are displayed in the title/address bar, whether a security alert is prompted for URLs with TLS errors (T5), and whether IABIs could defend against phishing URLs with a fake HTTPS lock icon (T6) and a long sub-domain name (T7). Third, during navigation of the web page, we test whether IABI could give a specific warning if the browsed page asks users to input passwords in a (potentially phishing) login form (T8).

Although our analysis focused on the apps' performance rather than a study with direct end users, our cross-platform analysis results show the following major security findings:

- About 30% of the tested apps do NOT display the complete URL, thus fail to provide enough information for a user to trustfully open an URL. Most of these apps omit the scheme (HTTP or HTTPS), while two apps (*Weibo* and *Quora*) completely hide the URL. Another 30% of the apps, despite outputting the full URL, display additional favicon or title information, which enables attackers to craft a fake favicon/title to mislead users.
- Nearly all custom IABIs have various problems in providing sufficient indicators to faithfully display an in-app page to users, whereas ten IABIs that are based on Chrome Custom Tabs or SFSafariViewController are generally secure. Specifically, among the 15 apps implementing their own IABIs, over half do not display the domain name in the address bar, and nearly none provide HTTP(S) indicators, which makes them fail to defeat phishing with a lock emoji or a long subdomain.
- Only a few IABIs, from the *QQ*, and *QQ Mail* apps, give specific warnings to remind users of dangerous operations (e.g., password inputting) during navigating a login page.

To understand developers' reaction to our findings and to potentially provide our recommendations on fixing severe IABI issues, we issued security reports to all affected apps (details in Section 5). Most developers acknowledged our findings and agreed with our assessment. In particular, *Instagram* had fixed its issues as we reported and *LinkedIn* would patch it in its future versions. However, we also found that developers' willingness and readiness to fix usability security issues are rather low compared to fixing technical vulnerabilities. Specifically, they refused to recognize them as *vulnerabilities* and were not willing to patch or improve their risky IABIs. Nevertheless, to help mitigate risky IABIs and guide future designs, we propose a set of secure IABI design principles in Section 6.

To sum up, we make the following contributions in this paper:

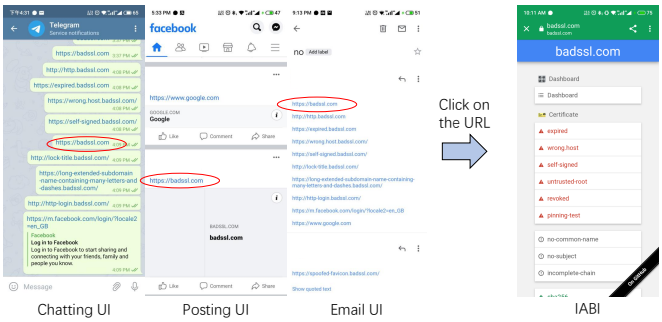

Figure 1: The process of opening a URL within an app. It demonstrates the 3 sample situations in which we want to open a URL within an app, including chatting with friends (Chatting UI), posting on the social network (Posting UI), and reading or sending an email (Email UI). When we click on the URL (e.g., https://badssl.com), an app may use in-app browsing interfaces to open the URL.

- *(Problem and analysis in §2-§3)* We summarized the attack surfaces on interacting with an IABI and performed a systematic analysis with eight security tests.
- *(Measurement results in §4)* We obtained cross-platform analysis results and their three major security findings by extensively testing 25 high-profile mobile apps.
- *(Reporting and defense in §5-§6)* We reported our findings to affected vendors and analyzed their responses. We further proposed a set of IABI design principles.

2 BACKGROUND

When we are using mobile apps to, e.g., chat with friends, read posts on social platforms, or read emails, we often need to open a URL link. In order to provide a "one-stop" service to keep users within the app interface without the need of switching to a web browser, many apps have implemented their own in-app browsing interfaces (IABIs) which typically use the underlying browsing engines to load the web content.

Figure 1 shows a typical process of opening a URL in an IABI. Sometimes the IABI may also contain an address bar to display the information related to the web page. Moreover, as shown in Figure 2, apps have three ways to handle an URL request when a user clicks the URL. First, some apps choose to jump out of the app and open a browser app to display the web page. This situation is out of the scope of our paper because they do not contain in-app browsing interfaces. The second way is to customize a browsing interface based on the Chrome Custom Tabs (CCT) [2] and SFSafariViewController (SF) [4] libraries. The third way is to create their own IABIs, which display the web page in the form of WebView (for Android) [1] or UIWebView (for iOS) [5] instances. Next, we explain the relevant terms in more details.

IABI (in-app browsing interface) and its address bar. We refer to the UI design of the entire screen when a mobile app opens a web page as the In-App Browsing Interface (IABI). In this paper, we mainly focus on the usability problems of IABIs' address bars.

Underlying browsing engines. The implementation of IABIs typically uses one of the following browsing engines:

- Chrome Custom Tabs (CCT in Android) [2, 24]

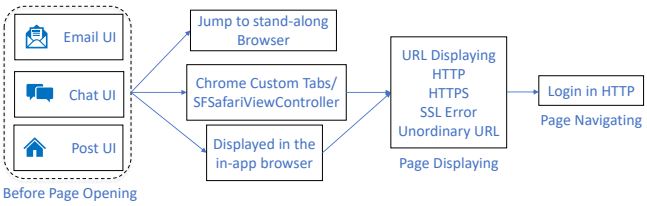

Figure 2: Three stages of interaction between an end user and the IABI and their potential usability security risks, including (i) usability trust given before users *open* a URL, (ii) security indicators to faithfully *display* an in-app page to users, and (iii) specific warnings to remind users of dangerous operations during *navigating* a login page.

- SFSafariViewController (SF in iOS) [4, 24]
- WebView (in Android) [1, 19]
- UIWebView (in iOS) [5, 27] and
- Custom browser engines implemented in native code [26].

Chrome Custom Tabs and SFSafariViewController. CCT is supported by Chrome, which is a web browser developed by Google. A mobile app can send a special intent to Chrome to launch a CCT to open websites without implementing a built-in browser engine by themselves, assuming that Chrome is installed on the smartphone. Chrome also provides well-encapsulated APIs for developers to make some limited browsing UI customization, such as color and animation. Similarly, iOS has SF supported by Safari for developers to incorporate into their apps easily.

WebView and UIWebView. Although CCT and SF provide convenient ways for a developer to implement IABI, one could choose, instead, to use lower-level display engines WebView and UIWebView (or even custom engines in native code [26]) for more comprehensive control over the UI. They allow developers to monitor specific events (e.g., loading and navigating) upon triggering of which developers can gather event information and make corresponding responses. The lower-level implementation of this design provides developers with very flexible control of the UI, which also implies more opportunities for design mistakes.

3 OVERVIEW OF OUR ANALYSIS

To reveal the usability security issues of IABIs in real-world applications, we analyze the IABIs in three phases corresponding to opening, displaying, and navigating a web page; see Figure 2. We design detailed security tests (T1 ~ T8) to reveal security properties of the design of IABIs in real-world applications. URLs tested include those provided by https://badssl.com/ and homepages of Google and Facebook.

(1) https://badssl.com
(2) http://http.badssl.com
(3) https://expired.badssl.com
(4) https://wrong.host.badssl.com
(5) https://self-signed.badssl.com
(6) http://lock-title.badssl.com
(7) https://long-extended-subdomain-name-containing-m any-letters-and-dashes.badssl.com
(8) http://http-login.badssl.com
(9) https://google.com
(10) https://m.facebook.com/login/

In this section, we will explain the details of our security tests T1 ~ T8 and then describe in detail how the individual tests are performed in each stage.

3.1 Analyzing Risks before Page Opening

In the first phase of our analysis, we investigate how an app displays URLs before users tap on them to open the web pages (**T1**). The simplest design is to display only the URL of the website without other content. Some apps may show a box below the URL with additional information about the web page, e.g., the title and favicon.

T1 is an indispensable step in the process of opening a web page in the app. Since it is not a part of the IABI, we cannot compare it with mobile browsers, and existing work does not provide any principles about its design. Therefore, we assigned a GOOD rating according to criteria of other tests (T2, T6 & T7, see Section 3.2). It could be counter-intuitive, but we find that only displaying the URL without any other information could actually be a GOOD design because any additional information displayed (e.g., favicon) could potentially be taken advantage of by an attacker to provide misleading information (e.g., favicon being a lock emoji). NEUTRAL and BAD ratings are awarded accordingly.

To perform the test, we input all tested URLs to the subject apps and check the corresponding display. Note that at this moment the app does not open the website yet. Some apps may pre-load the website to get brief information about the website. We discuss the results of T1 in Section 4.2.

3.2 Analyzing Risks on Page Displaying

After an end user taps on the URL, the app could open the web page by switching to a stand-alone web browser (out of scope of our paper) or within the app implemented either with Chrome Custom Tabs (CCT)/SFSafaraViewController (SF) or its customized IABI.

When displaying the corresponding web content, different apps could have their own design on the address bar, which is the focus of the analysis in this stage. It includes six tests as follows.

- **T2**: whether the URL is shown on the address bar (using URL9). End users need this information to know the origin of the page. Various designs include showing the URL and/or domain name.
- **T3 and T4**: how HTTPS and HTTP protocols are handled in the URL (using URL1 and URL2). The HTTPS indicator is very intuitive for users to recognize whether a web page meets the TLS requirement or not. IABIs should display the corresponding indicator in both HTTP and HTTPS webpages. If an IABI only displays the HTTPS indicators but shows no indicator for an HTTP page, the user may not know that the HTTP page is not secure before she opens another HTTPS page and sees the indicator.
- **T5**: how SSL errors are handled (using URL3~5). The SSL errors tested include expired certificates, wrong hosts, and self-signed certificates. Various designs include blocking access, prompting options to end users, or accessing the web page without any warnings.

- **T6**: how the title with a lock emoji is displayed (using URL6). Showing the lock emoji could mislead end users into believing that it is a secure web page with HTTPS protocol.
- **T7**: how URLs with long subdomain names are displayed (using URL7). Displaying only the long subdomain without the domain name could present an illusion of visiting a trusted domain.

Our ratings for T2 ~ T7 are based on the evaluation of security indicators and principles on mobile browsers in existing work [9–11], which perform systematic analysis based on best pratices outlined in the World Wide Web (W3C) guidelines [6]. In the following explanations, we first outline principles we extract from such guidelines that are applicable to IABIs, and then justify our definitions of the various ratings used in this paper.

1) **Identity Signal: Availability.** The security indicators showing identity of a website MUST be available to the user through either the primary or secondary interface at all times. We believe that IABIs should at least display the domain name of a website which is the basic identity of a page. Therefore in **T2**, we assign a GOOD rating to displaying the URL or domain name of a web page.
2) **Error messages: Interruption/Prceeding options/Inhibit interaction.** These three principles require that the error warnings MUST interrupt the users' current task and inhibit the user to interact with the destination website. Meanwhile, the warnings MUST provide the user with distinct options (MUST NOT be only to continue). Accordingly, in **T5** we test how IABIs react to erroneous certificates, like wrong-host, expired, or self-signed certificates. Our GOOD rating is consistent with this guideline, which is displaying a prompt with the option to continue or not before the user opens the SSL-error page. The only difference is that we relax the requirement a bit and allow IABIs to directly stop loading that page without providing options. A BAD rating is given to designs which directly open those pages with certificate errors. We assign NEUTRAL to designs that handover the issue to a standalone browser, which is "lazy" but not compromising security.
3) **TLS indicator: Availability.** The TLS indicators MUST be available to the user through the primary or secondary interface at all times. Accordingly, we conduct **T3** and **T4**, which test whether the HTTPS and HTTP indicators are displayed on the address bar. A GOOD rating is to display the indicators (lock/exclamation icons) to show whether a page meets the TLS requirement. A BAD design does not show any indicators. Displaying the scheme ("https://" or "http://") also serves as an indicator, although not as intuitive as icons and therefore receives a NEUTRAL score.
4) **TLS indicator: Content and Indicator Proximity.** Content MUST NOT be displayed in a manner that confuses hosted content or browser indicators. In this paper, we conduct **T6** and **T7** to test whether the lock emoji in the titles and long sub-domain names could confuse the users. According to this guideline, the IABIs should not allow lock emoji to mimic the HTTPS indicators or allow the long sub-domain name to mislead users, which are the GOOD ratings of T6 and T7. We define NEUTRAL as displaying both the security indicators and the lock emoji (T6). In T7, NEUTRAL is not displaying the domain name (though it is a BAD design in T2), and a BAD design refers to those ignoring this principle and allowing the two items to mislead users.

The results of our analysis into T2 ~ T7 are presented in Section 4.3.

3.3 Analyzing Risks on Page Navigating

When a user navigates a web page in an app's IABI, it is dangerous to input the username and password information in a login form because IABIs are more vulnerable to phishing attacks than standalone browsers [18]. Therefore, well-designed IABIs should give *specific* and *extra* warnings to remind users of the risk of inputting passwords during navigating a login page. Moreover, they should cover not only insecure HTTP login pages but also HTTPS login pages with vaild but illegitimate certificates. That is because an attacker can forge a phishing page with the title of a popular page (e.g., `alibaba.com`) by simply using a CA-issued (valid) certificate on a similar domain (e.g., `alibababa.com`) and thus meet the TLS requirement.

Based on this consideration, we test whether IABIs would show a *specific* or *extra* warning during navigation of a login page as compared with their normal behaviors on a non-login page (**T8**). We conduct this test using URL8 and URL10, which are example HTTP and HTTPS login pages, respectively. Note that for *Facebook* and *FB Messenger*, we use our university's HTTP login page since URL10 is already the Facebook login page. For each IABI, we navigate to the two login pages, input a username and a password, and check whether the IABI shows a specific warning on our operation.

Specifically, we assign a GOOD rating if the subject IABI provides specific warnings for both HTTP and HTTPS login pages. In contrast, we give the NEUTRAL and BAD ratings if the subject IABI displays a warning for at least one login page or has no such warning for both pages, respectively. We show the test results of T8 in Section 4.4.

4 CROSS-PLATFORM ANALYSIS RESULTS

We begin this section with an overview of the test apps used in our analysis (Section 4.1). Section 4.2, 4.3, and 4.4 cover our analysis results on IABIs handling of URLs before web page opening, during web page opening, and during web page navigation, respectively.

4.1 Test Apps and Overall Analysis Results

Table 1 shows the category, number of ratings on Apple Store, and installs on Google Play of the 25 high-profile applications we use in our analysis. We reiterate that these apps are selected because they have an IABI and they are good representations of popular apps from various usage categories. In each category, we sort these apps according to their popularity on Apple Store because users in China typically do not use Google Play to install Android apps.

Table 2 presents an overview of our analysis results, with details discussed in Section 4.2, 4.3, and 4.4. Note that when an app uses Chrome Custom Tabs (CCT) or SFSafariViewController (SF) to implement IABI, its behavior will always be the same (defined in CCT and SF; see row "CCT/SF" in Table 2). Therefore, our analysis and discussion below focus more on the analysis of "own IABI" implementation, where each subject app makes its own call on the

Table 1: Subject mobile apps we tested and # of their Apple Store ratings and Google Play installs.

Category	App Name	# of Ratings	# of Installs
Chat	WeChat	4,000,000+	100M-500M
	FB Messenger	1,000,000+	1B-5B
	QQ	700,000+	5B-10B
	Snapchat	300,000+	1B-5B
	LINE	200,000+	500M-1B
	Telegram	60,000+	100M-500M
	KakaoTalk	60,000+	100M-500M
	Hangouts	40,000+	1B-5B
Social	Instagram	10,000,000+	1B-5B
	Weibo	500,000+	1B-5B
	Facebook	400,000+	5B-10B
	Twitter	200,000+	500M-1B
	Tumblr	200,000+	100M-500M
	VK Russia	200,000+	100M-500M
Email	Gmail	100,000+	5B-10B
	163 Mail	100,000+	50M-100M
	Mail.ru	100,000+	50M-100M
	QQ Mail	50,000+	100M-500M
Business	Alipay	600,000+	1B-5B
	LinkedIn	50,000+	500M-1B
News	Toutiao	2,000,000+	100M-500M
	Reddit	1,000,000+	1M-50M
	Baidu	800,000+	1M-50M
	Zhihu	700,000+	100M-500M
	Quora	70,000+	1M-50M

design and implementation. As the results on Android and iOS are the same in most subject apps, in the following detailed discussions, we will first focus on our analysis on the Android platform followed by a brief comparison with results on iOS.

4.2 Usability Risks before Page Opening

As discussed in Section 3.1, this part of the analysis concerns how the URL is displayed before end users opens it.

4.2.1 T1: Displayed URLs before page opening. Figure 3 provides screenshots of a few representative apps on how they display a URL before end users click on it. After analyzing the different handling, we categorize them into three buckets — Good, Neutral, and Bad[1].

GOOD. The most common way (accounts for roughly 50% of our subject apps) is displaying the complete URL; see Case 1 in Figure 3. We consider this a GOOD practice as end users can see the full URL without being misled by maliciously crafted favicons or titles (see the BAD cases later).

NEUTRAL. Some apps may display additional information of the corresponding web page (see Case 2, 3, and 4 in Figure 3; we confirm that they show the complete URL even if the URLs contain sub-domains and additional paths) including title, domain, favicon, and even some page content. Although such additional information might help in a legitimate URL, it could be also misleading in other cases. For example, a fake lock favicon is displayed in the case of *LINE* in Figure 3. That said, apps in this category also display the complete URL for inspection by end users; we therefore consider the

[1]We notice cases in which a subject app displays URLs in different ways in different activities (e.g., in a chat window and in a wall/post window), in which our analysis result is based on its worst-case categorization.

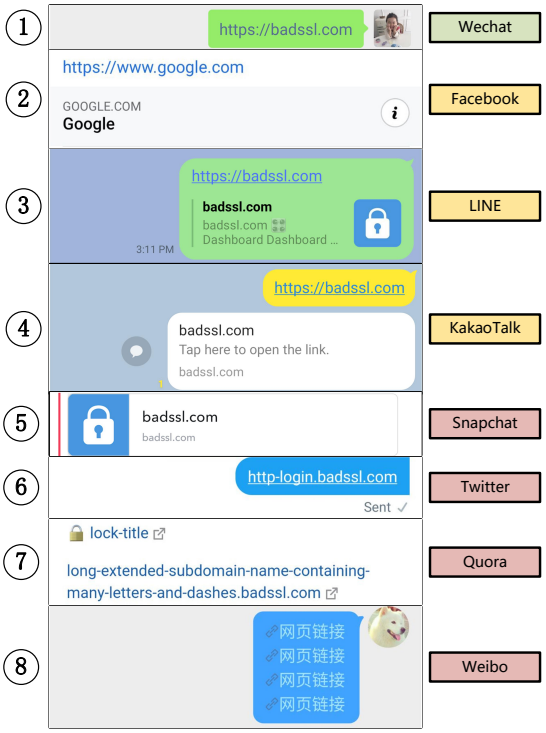

Figure 3: Examples of displayed URLs before page opening (T1).

practice of displaying the complete URL together with additional (potentially misleading) information as NEUTRAL.

BAD. This category refers to apps that do not display the complete URL (including cases with missing HTTP/HTTPS scheme). For example, *Snapchat* (Case 5) only displays the title, favicon, and domain name (and see the lock-looking favicon). It presents an example in which end users are given no information about the actual security of the URL. *Twitter* and *Quora* (Case 6 and 7) have the scheme of the URL stripped off with only the domain name left. *Weibo* (Case 8) displays an identical label for every URL, which is also BAD in failing to provide adequate information of the URL.

iOS. Most of the apps perform exactly the same on Android and iOS in this analysis, with the exception of *VK Russia* whose Android version displays only the complete URL (GOOD) while its iOS version displays the title and domain name as well (NEUTRAL).

Takeaway in §4.2: About 30% of the subject apps do NOT display the complete URL, failing to provide the necessary security indicators. Most of them omit the scheme (HTTP or HTTPS), while two apps completely hide the URL content. Another 30% of the apps, despite outputting the full URL, display additional favicon or title information, which potentially enables attackers to maliciously craft a fake favicon/title to mislead end users.

4.3 Usability Risks on Page Displaying

As discussed in Section 3.2, this part of the analysis focuses on the title and address bar developers typically add to enhance the user interface of the app. We first examine the display of URL when

Table 2: Cross-platform IABI test results in terms of their usability risks in opening, displaying, and navigating a web page.

Category	App Name	CCT/SF	Own IABI	Before opening	Detailed test results when the subject app uses its own IABI						
					Displaying a web page						Navigating
				T1: URL	T2: URL	T3: HTTPS	T4: HTTP	T5: SSL	T6: Lock-title	T7: Sub-domain	T8: Login
CCT/SF		* | *	* | *	* | *	✔| ✔	✔| ✔	✔| ✔	✔| ✔	◑| ✔	✔| ✔	✗| ◑
Chats	WeChat	✗| ✗	✔| ✔	✔| ✔	✗| ✗	✗| ✗	✗| ✗	✔| ✔	✗| ✗	◑| ◑	✗| ✗
	FB Messenger	✗| ✗	✔| ✔	◑| ◑	✔| ✔	✔| ✔	✗| ✗	✔| ✔	✗| ✔	✗| ✔	✗| ✗
	QQ	✗| ✗	✔| ✔	✔| ✔	✗| ✗	✗| ✗	✗| ✗	✔| ✔	✗| ✗	◑| ◑	✔| ✔
	Snapchat	✗| ✗	✔| ✔	✗| ✗	✔| ✔	✗| ✔	✗| ✗	✔| ✔	✗| ✗	✗| ✗	✗| ✗
	LINE	✗| ✗	✔| ✔	◑| ◑	✔| ✔	◑| ◑	◑| ◑	✔| ✔	◑| ◑	✔| ✔	✗| ✗
	Telegram	✔| ✔	✗| *	✔| ✔	-| -	-| -	-| -	-| -	-| -	-| -	-| -
	KakaoTalk	✗| ✗	✔| ✔	◑| ◑	✔| ✔	✔| ✔	✗| ✗	✔| ✔	✔| ✔	✗| ✗	✗| ✗
	Hangouts	✔| ✔	✗| *	✔| ✔	-| -	-| -	-| -	-| -	-| -	-| -	-| -
Social	Instagram	✗| ✗	✔| ✔	◑| ◑	✔| ✔	✗| ✗	✗| ✗	✔| ✔	✗| ✗	✗| ✗	✗| ✗
	Weibo	✗| ✗	✔| ✔	✗| ✗	✗| ✗	✗| ✗	✗| ✗	✔| ✔	✗| ✗	◑| ◑	✗| ✗
	Facebook	✗| ✗	✔| ✔	◑| ◑	✔| ✔	✔| ✔	✔| ✔	✔| ✔	✔| ✔	✗| ✗	✗| ✗
	Twitter	✔| ✔	✔| *	✗| ✗	(o)✔| -	(o)◑| -	(o)◑| -	(o)◑| -	(o)◑| -	(o)✗| -	(o)✗| -
	Tumblr	✔| ✔	✗| *	✗| ✗	-| -	-| -	-| -	-| -	(t)✔| -	-| -	-| -
	VK Russia	✔| ✔	✗| *	✔| ◑	-| -	-| -	-| -	-| -	-| -	-| -	-| -
Email	Gmail	✔| ✔	✗| *	✔| ✔	-| -	-| -	-| -	-| -	-| -	-| -	-| -
	163 Mail	✗| ✗	✔| ✔	✔| ✔	✗| ✗	✗| ✗	✗| ✗	✔| ✔	✗| ✗	◑| ◑	✗| ✗
	QQ Mail	✗| ✗	✔| ✔	✔| ✔	✗| ✗	✗| ✗	✗| ✗	✔| ✔	✗| ✗	◑| ◑	✔| ✔
	Mail.ru	✔| ✔	✗| *	✔| ✔	-| -	-| -	-| -	-| -	-| -	-| -	-| -
Business	Alipay	✗| ✗	✔| ✔	✔| ✔	✗| ✗	✗| ✗	✗| ✗	✗| ✔	✗| ✗	◑| ◑	✗| ✗
	LinkedIn	✔| ✗	✔| ✔	◑| ◑	(o)✗| ✔	(o)✗| ✗	(o)✗| ✗	(o)✔| ✔	(o)✗| ✗	(o)◑| ✗	(o)✗| ✗
News	Toutiao	✗| ✗	✔| ✔	✗| ✗	✗| ✗	✗| ✗	✗| ✗	✔| ✔	◑| ◑	◑| ◑	✗| ✗
	Reddit	✔| ✔	✔| *	✔| ✔	(o)✔| -	(o)✔| -	(o)✗| -	(o)✔| -	(o)✔| -	(o)✗| -	(o)✗| -
	Baidu	✗| ✗	✔| ✔	✔| ✔	✗| ✗	✗| ✗	✗| ✗	✔| ✔	◑| ◑	◑| ◑	✗| ✗
	Zhihu	✗| ✗	✔| ✔	✗| ✗	✗| ✗	✗| ✗	✗| ✗	✗| ✔	✗| ✗	✗| ✗	✗| ✗
	Quora	✔| ✔	✗| *	✗| ✗	-| -	-| -	-| -	-| -	(t)✔| -	-| -	-| -

Column "CCT/SF": whether the subject app uses Chrome Custom Tabs (Android) or SFSafariViewController (iOS) to implement IABI.
Column "Own IABI": whether the subject app uses its own implementation of IABI. Note that an app could present both CCT/SF and Own IABI behavior (when the phone has and does not have Chrome/Safari installed, respectively). Since Safari is installed by default on iOS, the app's "own IABI" status is unknown when it already has SF implementation.
Row "CCT/SF": analysis results when using CCT/SF to implement IABI. Note that all subject apps will present the same behavior when using CCT/SF.
Cell format: <analysis result on Android> | <analysis result on iOS>
Symbols: ✔= Good or Yes; ◑= Neutral; ✗= Bad or No; '*' = Unknown; '-' = Same as CCT/SF; '(o)' = Using **(o)**wn IABI if no Chrome; '(t)' = CCT without **(t)**itle.

loading the web page (T2), and then test 4 types of URLs with HTTP (T3), HTTPS (T4), SSL errors (T5), and special cases (T6 and T7).

4.3.1 T2: Displayed URLs during page opening. The importance of proper display of URLs here is similar to that in T1, with three notable differences. First, T2 focuses on the display of URL only and leaves the analysis of scheme indicators to T3~T5. Second, web page redirection places an additional demand on the display of URLs while pages are being opened. Third, a preview of the page no longer adds any usability or functionality since the actual page is now opened. Results on representative apps are shown in Figure 4.

Chrome Custom Tabs and SFSafariViewController. We first take a look at how full-fledged browsers handle URL displays. Both Chrome Custom Tabs (CCT) and SFSafariViewController (SF) always display the domain name in their address bars, which is considered a GOOD design. Customization of CCT allows showing or hiding the title (using API `setShowTitle(true)`); see Case 1 and 2 in Figure 4). 10 of the 25 subject apps use CCT while 9 of them use SF.

We now turn our attention to IABI with each application's own design and implementation without using CCT/SF. Note that not all subject apps choose to provide their own IABI implementation;

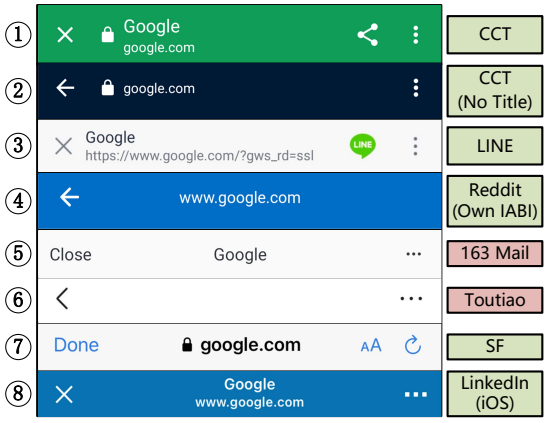

Figure 4: Examples of displayed URLs when a page is loaded (T2).

see column "Own IABI" in Table 2. Similar to what we did with T1, we categorize these implementations into buckets of GOOD, NEUTRAL, and BAD, and present our findings on the Android

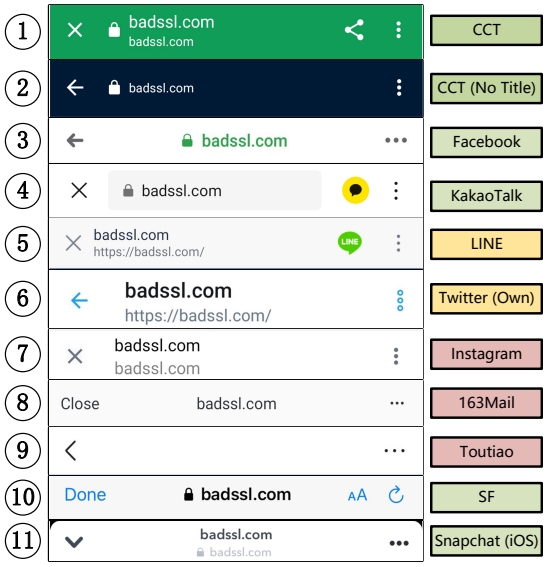

Figure 5: Examples of HTTPS indicators (T3). '(Own)' refers to the app's own IABI, similarly hereinafter.

platform first followed by a comparison with corresponding iOS apps.

GOOD. We consider it a GOOD design provided that either complete URL or its domain is displayed. A total of 8 Android apps satisfy this requirement (e.g., Case 3 in Figure 4) out of 18 subject apps[2] that provide their own IABI implementations.

BAD. The other 10 subject apps only display the title of the web page without the URL (or domain name), e.g., *163Mail* (Case 5), or even no title/address bar at all, e.g., *Baidu*, which is a BAD design for the same reasons discussed in T1. An interesting observation is that 9 out of these 10 subject apps are from China.

Page redirection. Web page redirection is common, and IABIs shall always display information of the URL of the final landing page being opened. Our evaluation shows that all the subject apps pass this test, should they use `WebView.getUrl()` to directly retrieve the URL or use the arguments of a set of hook functions within `WebViewClient` (`onPageStarted()` and `onPageFinished()`).

iOS. On iOS, LinkedIn displays the title and domain name on the address bar (Case 8), which is GOOD (as opposed to its BAD design on the Android counterpart). Other apps have exactly the same performance on the two mobile platforms.

4.3.2 T3: HTTPS Indicators. Apps typically provide HTTPS indicators in the form of text ("https" in the URL) or a lock icon. Screenshots of representative apps in this analysis are shown in Figure 5.

Chrome Custom Tabs and SFSafariViewController. Both CCT and SF use a lock icon as the indicator of HTTPS, which is not customizable or removable by the app developers (Case 1 and 2 in Figure 5). We consider them GOOD designs.

GOOD. Similar designs can be found in own IABI implementations in three apps, *Facebook* (Case 3), *FB Messenger* and *KakaoTalk* (Case 4).

[2]Three apps (*Twitter*, *LinkedIn* and *Reddit*) provide both CCT/SF and own IABI implementations.

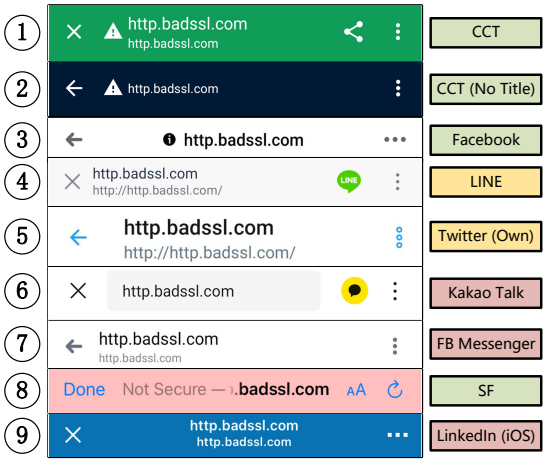

Figure 6: Examples of HTTP indicators (T4).

NEUTRAL. Some apps rely on the scheme portion of the full URL (the text "https") to indicate that HTTPS protocol is being used (Case 5 and 6), which might not be very intuitive but suffice for advanced users.

BAD. The absence of any HTTPS indicators (text or lock icon) is a BAD design. Surprisingly, we found 12 out of the 18 subject Android apps with their own IABI implementation falling into this category, including *Instagram* developed by the Facebook company (*Facebook* and *FB Messenger* are GOOD though).

iOS. The iOS version of *Snapchat* displays a lock icon in its own IABI implementation, while its Android version does not have any indicators. Other apps have identical behavior with respect to T2 on Android and iOS.

4.3.3 T4: HTTP Indicators. As discussed in Section 3.2, proper indicators for HTTPS should not exempt an app from displaying an HTTP indicator/prompt. In other words, an HTTP indicator should always be displayed regardless of the presence or absence of HTTPS indicators. Figure 6 shows screenshots of representative apps in this analysis.

Chrome Custom Tabs and SFSafariViewController. CCT uses an exclamation mark icon in place of the lock icon when the URL does not meet TLS requirements; see Case 1 and 2 in Figure 6. This icon is very intuitive and can give the user a clear warning. Same as the lock icon for HTTPS, this exclamation mark icon cannot be customized or removed by a developer. SFSafariViewController uses the text "Not Secure" as the indicator; see Case 8. Both of them scored GOOD in this test.

GOOD. Similar to the analysis of HTTPS indicators, a GOOD design should always display an insecure indicator for HTTP. Unfortunately, *Facebook* is the only app scoring GOOD design in this test (Case 3).

NEUTRAL. Displaying the complete URL with the scheme portion of text "http" also serves the purpose with a less intuitive interface, and is considered a neutral design in our analysis. *LINE* and *Twitter* join this category (Case 4 and 5).

BAD. The absence of any HTTP indicator is considered a BAD design, and we have 15 out of 18 apps with their own IABI implementation in this category including *FB Messenger*. This is an alarming finding.

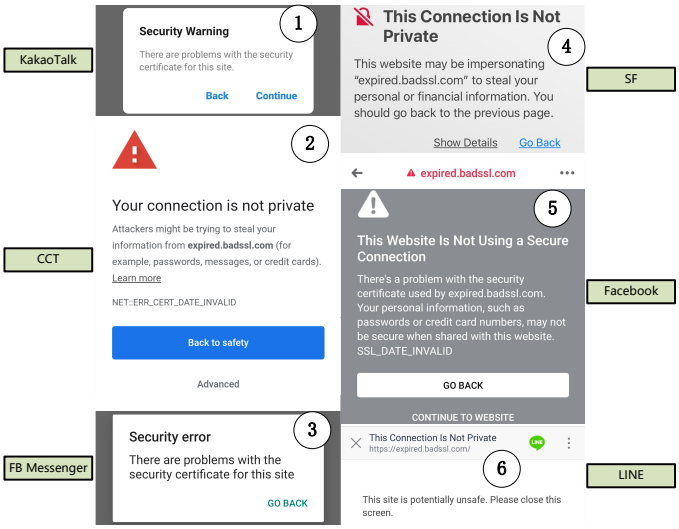

Figure 7: Examples of displaying expired certificates (T5).

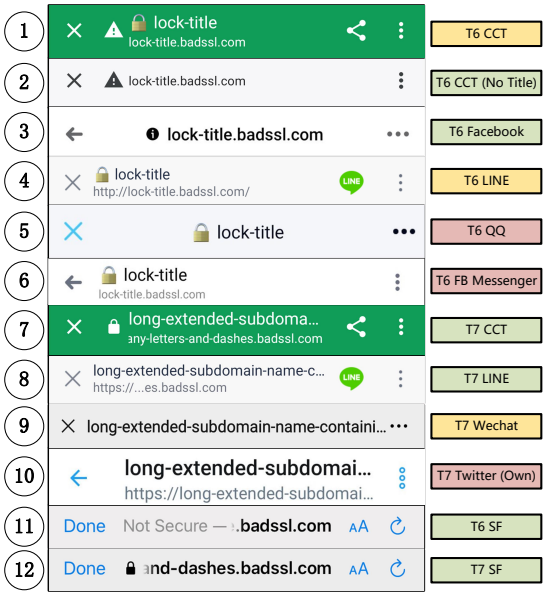

Figure 8: Examples of displaying special URLs (T6 & T7).

iOS. Similarly, *Facebook* is the only app with GOOD design here with respect to HTTP indicators on iOS apps.

4.3.4 T5: Certificate Errors. As discussed in Section 3.2, an app is supposed to inform end users upon certificate errors (with expired or self-signed certificates, or with certificates of the wrong host). We test all subject apps with URLs that contain such certificate errors, and examine their corresponding prompts; see Figure 7 with expired certificates as examples.

Chrome Custom Tabs and SFSafariViewController. Since most layman end users do not possess the necessary technical background to make informed decisions when they are prompted with a certificate error, both CCT and SF (and the corresponding full-fledged browsers) introduce "twisted" routes for end users to proceed opening web pages with certificate errors. Cases 2 and 4 in Figure 7 show examples of the design of such "twisted" routes, where an end user will have to choose "Advanced" or "Show Details" before they are given the option to continue browsing. We consider these GOOD designs with best security usability practice.

That said, both CCT and SF choose to remember such end user decisions *across all apps with CCT/SF implementation*. In other words, an app with CCT/SF implementation would skip the certificate error warning if a user had chosen to proceed with browsing the same URL on any other CCT/SF apps. Although the CCT/SF implementation is generally considered GOOD, such a design choice sacrifices security for usability.

GOOD. Here we relax the security requirement and consider apps that either refuse to open the page or prompt end users with various options as GOOD designs.

With such relaxation on the definition of GOOD designs, all but 3 subject apps with their own IABI implementations are in this category. Among them are 7 apps that simply refuse to open the page, and 8 apps that prompt the end users and give options to proceed. There are two interesting observations worth noting when we dig deeper into the latter group.

First, 3 out of the 8 apps show details of specific certificate errors (e.g., *Facebook*, Case 5) while the other 5 apps skip the details of errors (e.g., *KakaoTalk*, Case 1). Second, all these apps remember the user selection and would not display any SSL error indicators after proceeding to open the web page, except *Facebook* (case 5) which turns the domain name into red color and include a red exclamation mark icon even after end user chooses to proceed.

NEUTRAL. *Twitter* instead chooses to launch the system default browser to handle the web pages with certificate errors. We consider this design acceptable (NEUTRAL) although the burden is now shifted to the default browser.

BAD. Ignoring the certificate errors or directly opening the insecure web page are both considered as BAD designs. In this category we have *Alipay* which directly opens a web page with wrong host certificates and *Zhihu* which indiscriminately displays a prompt of visiting external websites regardless of presence of absence of certificate errors.

iOS. Surprisingly, all iOS subject apps deliver GOOD designs including *Alipay* and *Zhihu* whose Android versions are BAD. Both two iOS apps show blank pages when attempting to open pages with certificate errors. We suspect that this better behavior of iOS apps is due to stricter control for certificate errors on the iOS platform.

4.3.5 T6 & T7: Special URLs. As discussed in Section 3.2, T6 and T7 concern about special URLs where a lock emoji is part of the title, and where extended sub-domain names are used, respectively. Figure 8 shows screenshots of representative apps when they process such special URLs.

Chrome Custom Tabs and SFSafariViewController. CCT could be configured with or without the title of the page being visible. When the title is visible (Case 1 of Figure 8), the lock emoji (part of the title) and the exclamation mark icon (due to the HTTP protocol being used) show up next to each other, which is confusing but not too bad as the warning is there. In absence of the page title,

either due to developer configuration in CCT (Case 2) or due to SF in use (Case 11), the HTTP warning is there without any confusion.

Regarding T7, both CCT and SF display the *suffix* of the URL (with the long subdomain trimmed), which means that both CCT and SF are immune to T7 attacks (see Cases 7 and 12).

GOOD. Regarding T6, a potential GOOD design we'd suggest is to detect the use of lock (or similar) emojis in the title and replace them with unambiguous text or symbols. Unfortunately, none of the subject apps has a similar design (not even the CCT/SF implementations). Therefore, we only consider their own IABI implementations not showing the title as GOOD designs. Except for those CCT apps which choose not to show the title, only *Facebook* (case 3 in Figure 8) and *KakaoTalk* have a GOOD design in this test.

In T7, we consider designs that prioritize the display of domain name over subdomain name GOOD. *LINE* is the only GOOD one in our tests (Case 8).

NEUTRAL. When the lock emoji in the title is shown, we consider it NEUTRAL if either the full URL or an HTTP indicator (e.g., exclamation mark icon) is also shown, so that end users still have a way of telling that the page is insecure. Examples include CCT implementation, *LINE* (Case 4), and *Twitter*.

In evaluating T7, we find that 9 of 18 apps with their own IABI implementations only display the title of the page without the URL (e.g., *WeChat* in Case 9), which strictly speaking does not fall short on the extended subdomain but is still misleading. We therefore categorize this as NEUTRAL.

BAD. In T6, 11 apps display the lock emoji without any HTTP indicator or displaying the complete URL (e.g., Cases 5 and 6). End users have a high chance of being misled by the lock emoji, and we consider such designs BAD. Regarding T7, 8 of the subject apps display the subdomain name with the domain name missing (Case 10).

iOS. Again, most apps have similar behavior on their Android and iOS version, with the exception of *FB Messenger* iOS app which only has the domain name on its address bar and is considered as a GOOD design for T6. For T7, *FB Messenger* iOS app displays both the head and the tail of the domain name, which is also a GOOD design. *LinkedIn* iOS app, on the other hand, suffers with displaying the subdomain (BAD) although its Android counterpart only shows the title (NEUTRAL).

Takeaways in §4.3:

- **Ten apps using Chrome Custom Tabs or SF perform GOOD on nearly all the tests from T2 to T7.**
- **More than half of the remaining 15 apps do not display the domain name in their own IABI implementation, and nearly none of them provides HTTP and HTTPS indicators. This makes it difficult for end users to differentiate between secure and insecure pages. Fortunately, all apps behave GOOD when handling URLs with certificate errors.**
- **Moreover, nearly all those 15 apps have BAD performance on handling a title with the fake lock emoji or a long subdomain name, which could lead to insecure pages being misinterpreted as secure ones.**

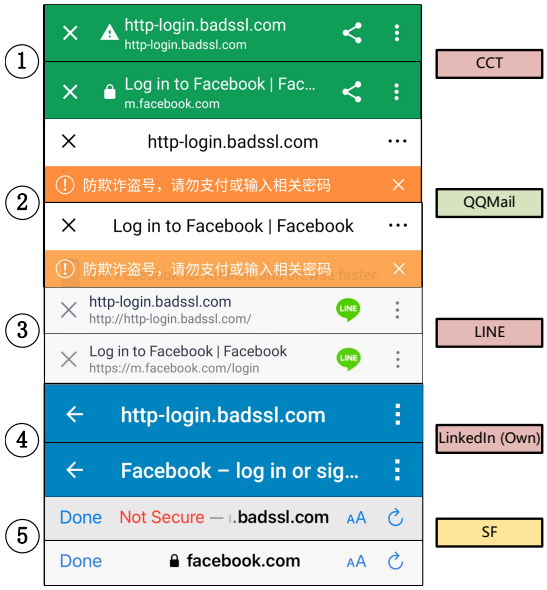

Figure 9: Examples of whether displaying specific warnings in the title bar when users browse a (potentially phishing) login page (T8). We tested both HTTP and HTTPS login pages.

4.4 Usability Risks on Page Navigation

As mentioned in Section 3.3, our last test (T8) is to see how a subject app's IABI implementation reacts to dangerous operation (e.g., password inputting) during the navigation of a web page. Specifically, T8 tests whether IABIs would show a *specific* or *extra* warning on a login page as compared with their normal behaviors on a non-login page. Figure 9 shows the screenshots of some representative IABIs.

Chrome Custom Tabs and SFSafariViewController. As shown in Case 1 of Figure 9, CCT does not provide any additional warning for password inputting when navigating both HTTP and HTTPS login pages. That said, it has the same behaviors as previously in T3 and T4; see Figure 5 and 6. Therefore, we consider CCT's rating as BAD in this test. In contrast, SF performs slightly better than CCT. As shown in Case 5 of Figure 9, SF highlights the display of "Not Secure" (normally displayed for all HTTP pages; see Case 8 in Figure 6) by changing its color to red when end users navigate an HTTP login page, which is more striking for end users to notice. However, SF does not have specific warnings on HTTPS login pages. As a result, we assign NEUTRAL to SF's performance on T8.

GOOD. According to our specification in Section 3.3, we consider those that provide specific warnings for both HTTP and HTTPS login pages as GOOD designs. *QQ Mail* (Case 2) and *QQ* (both of which are developed by Tencent) display such prompt when the user enters username or password into both HTTP (URL8) and HTTPS (URL10) web page, which are GOOD.

NEUTRAL. Some apps display an extra warning (other than the normal HTTP indicator) during navigation of an HTTP login page but fail to provide any warning on an HTTPS login page. According to our specification in Section 3.3, we rate these apps'

IABIs as NEUTRAL. For all tested apps, we find that only the apps using SF have such performance.

BAD. The other apps score BAD in T8, including *WeChat* which is also developed by Tencent but they do not show any warnings in this test. Upon further investigation, we find that this anti-fraud tip which is used by the other two apps can be removed in *WeChat* if the domain name is registered on the developer platform of *WeChat*. So maybe some developers have registered this domain name on that platform so the prompt is removed.

iOS. iOS apps their with own IABI implementation behave the same as their Android versions in this test.

Takeaway in §4.4: Most of the tested IABIs do not provide specific warnings to remind users of the risk of inputting passwords during navigating a login page, regardless it uses HTTP or HTTPS. Even CCT/SF does not perform well in this teszhoumot.

5 APP DEVELOPERS' RESPONSES

To understand developers' reaction on our findings and to potentially provide our recommendations on fixing severe IABI issues, we issued security reports to all affected apps including *WeChat, FB Messenger, QQ, Snapchat, Instagram* (fixed as we reported), *Weibo, 163 Mail, QQ Mail, Alipay, LinkedIn, Toutiao*, and *Baidu*, through their bug bounty programs or security contact emails[3]. Most of the apps acknowledged our findings and agreed with our assessment, but refused to recognize them as *vulnerabilities*, i.e., they consider the reported issues out of the scope of their bug bounty programs. By analyzing their responses in detail as follows, we find that developers' willingness and readiness to fix usability security issues are rather low compared to fixing technical vulnerabilities, which is a puzzle in usability security research.

Facebook's response. While the *Facebook* app performs well in nearly all the tests, the other two apps from the same company, namely *FB Messenger* and *Instagram*, did not use the same IABI design and failed in our tests T3~T4 and T6~T8 (see §4). When we prepared the security reports for these two apps on 4 February 2021, we found that the latest version of *Instagram* had changed its IABI design to display a lock icon and an exclamation mark to indicate the HTTPS and HTTP pages, respectively. This suggests that they also noticed this problem (before our reporting) and made an improvement. Therefore, we focus on the response to our reports to *FB Messenger*.

There are two key points in the response given by the Facebook security team. First, they appreciated our report but said that our report does not qualify for their bug bounty program due to the social engineering nature of our reported attacks. Second, they already have a URL detection system called Linkshim, which could detect potentially malicious URLs and thus defend against IABI attacks. While we agree that the reported IABI usability issues would not cause the same security consequences as in the technical WebView vulnerabilities (e.g., [16, 19, 26, 27, 30]), the usability weaknesses in *FB Messenger*'s IABIs we demonstrated in §4 definitely give attackers ample rooms to successfully launch phishing attacks. Moreover, it is a puzzle to see that different apps developed by the same company make vastly different security decisions for seemingly the same component.

Snapchat's response. Compared with *Facebook*, the security team of *Snapchat* is more concerned on IABIs' usability security issues with the following response (despite that a priority fix would not be issued as similar to *FB Messenger*): "*While this attack scenario is quite interesting, we would consider this to be more of a defense-in-depth issue, rather than a discrete security vulnerability in Snapchat itself. Per our program rules, we generally don't accept issues pertaining to 'Reports solely indicating a lack of a possible security defense'. While we appreciate your suggestions here, we don't feel that this poses a severe enough risk to warrant a priority fix, and as such we'll be closing this report as Informative. We do appreciate your efforts here, and we hope you'll continue reporting security issues to us in the future.*"

LinkedIn's response. Compared to *Facebook* and *Snapchat*, the response from *LinkedIn*'s security team is more positive. Specifically, they consider patching it in the future versions of the *LinkedIn* app: "*Thank you for your detailed email and report, we appreciate it. We regularly review incoming reports to identify opportunities to improve our member experience and their safe interactions on the platform. We have taken note of these items and included it for consideration in our future roadmaps. Once again, we appreciate the detail and effort that went into this research & report.*"

Some Chinese IT companies' responses. We also received responses from a few large Chinese IT companies, including Tencent (developing *QQ, WeChat*, and *QQ Mail* apps), Alibaba (developing *Alipay*), ByteDance (developing *Toutiao*), Sina (developing *Weibo*), NetEase (developing *163 Mail*), and *Baidu*. Most of them did not take our security reports seriously — either indicating that it was a known problem (by Tencent) or simply closed our reports without an explanation (by Sina). The only exception goes to ByteDance, whose security team responded that they just followed "the industry's standard" but will report this issue to their product team.

All these responses suggest a play down from app developers when it comes to IABI usability security concerns.

6 SECURE IABI DESIGN PRINCIPLES

In this section, we propose a set of secure IABI design principles and corresponding code-level implementations in this section to help mitigate risky IABIs and guide future designs. Here we provide implemenations in Android as examples, and iOS developers can use corresponding counterparts in iOS.

Firstly, we recommend the use of Chrome Custom Tabs and SFSafariViewController for their good performance in our tests presented in this paper except for T8. The implementation guides are CCT [3] and SF [4]. Comparing to building one's own IABI, CCT/SF are easy to incorporate with little effort while achieving outstanding security design and optimized loading speed.

However, CCT and SF also have their limitations. CCT failed to provide an extra prompt to alert users when entering passwords on the web page. And they can only provide some basic customization options while developers may need deeper customization to fit in with their apps. In some region, Chrome (and therefore CCT) is not available, e.g., in the mainland China market.

[3] Apps using CCT/SF are generally secure; so we skip them. We failed to locate any feedback channel for *KakaoTalk* and *Zhihu* and therefore have to skip them as well.

Considering these shortcomings of CCT/SF, we propose the following IABI design principles for those developers who need their own IABI implementation. We devide them into three parts: design principles before opening the URL, on page displaysing and on page navigating. Considering these shortcomings of CCT/SF, we propose the following IABI design principles for developers who need their own IABI implemenation. We devide them into three parts: design principles before opening the URL, on page displaysing, and on page navigating.

(1) **Before Opening the URL:** In the chatting, posting, email UI and other possible UI that display the clickable URL, IABIs:
   (a) ***SHOULD display the complete URL and corresponding indicators of URL schemes.*** It would be GOOD that the indicator be more intuitive and eye-catching than title and favicon.
   (b) ***SHOULD NOT display any extra pre-loading information*** (e.g., favicion and title), unless that URL can be trusted.

(2) **On Page Displaying.** After the user taps the URL, developers could adopt the following five principles to better avoid the potential usability security issues on page displaying:
   (a) ***SHOULD display the full URL in the address bar*** to show the page origin. Developers can get the URL of the current page through `WebView.getURL()` or the arguments in the event handlers of `WebViewClient` (e.g., `onPageFinished`, shown in Listing 1).
   (b) ***SHOULD display the HTTP and HTTPS indicators***, which are intuitive for users to identify insecure web pages, potentially in conjunction with the scheme text in the URL. To this end, developers can override the `onPageFinished` method [7]. Here we provide a simple example in Listing 1.

```
public void onPageFinished(WebView view, String
     url){
  ...
  //Display url on the title bar.
  addressBar.setText(url);
  //obtain the scheme
  String scheme = URL(url).getProtocol();
  if(scheme.equals("https")){
    /*Display the HTTPS indicator*/
  }else{
    /*Display the insecure indicator*/
  }
  ...
}
```

Listing 1: Display the URL and indicators.

   (c) ***SHOULD NOT directly open URLs with certificate errors.***
      (i) Show a prompt, like a dialog box or a special page, which informs end users about SSL errors.
      (ii) Provide end users with the option to continue opening the URL in a covert manner, e.g., as in CCT which only shows the continue option after clicking on the "Advanced button" (Case 2 in Figure 7).

   To handle the certificate errors in the WebView, developers can override the event handler `WebViewClient.onReceivedSslError` [8], and show a dialog to inform the user about the error.

   (d) ***SHOULD handle the lock emoji in the title with extra care*** by:
      (i) Replacing it with the text to avoid misinterpreting as the HTTPS indicator; or
      (ii) Disallowing emoji; or
      (iii) Avoiding displaying the title.

   Developers can override the `onPageFinished` method, obtain the title of the web page by `WebView.getTitle()`, and detect the emoji code in the title. For example, the Unicode of the lock emoji is *U+1F512*. Another choice is to disallow all the unicode in the title. We do not recommend it as it will greatly damage the user experience.
   (e) ***SHOULD handle the long subdomain name with extra care.*** by:
      (i) Providing scrolling capability for end uses to read the complete domain name; or
      (ii) Prioritizing the display of domain name over subdomain name.

   To scroll the `TextView` (displaying the URL/domain name)in Android apps, developers can set its attribute in the layout xml file: `android:ellipsize=''marquee''`.
   To Prioritize the domain name, developers can set the attribute to: `android:ellipsize=''start''`.

(3) **On Page Navigating.** When the user tries to enter the password or other sensitive information in an web page, IABIs:
   (a) ***SHOULD show an additional warning regardless of HTTPS or HTTP pages.*** Case 2 in Figure 9 is a good example.
   To detect the input box of the password and username, developers can utilize the interaction between the Java and JavaScript code, i.e., using `WebView.loadUrl()` to execute JavaScript code to detect the 'password' type within the web page and give a corresponding prompt. An example is shown in Listing 2.

```
webView.loadUrl(
"javascript:(function(){" +
  "var objs=document.getElementsByTagName(\"input\");"
+ "for(var i=0;i<objs.length;i++)" +"{"
  + "var type = objs[i].getAttribute(\"type\");"
    + "if(type==\"password\"){"
      + "objs[i].onfocus=function(){"
        + "PROMPT_TO_ENTERING_PASSWORD"
       +"}"
    + "}"
   +"}"
 + "})()");
```

Listing 2: Display a prompt when entering passwords.

7 DISCUSSION

In this section, we discuss threats to the validity of our study and limitations. Specifically, the major threats are that we did not conduct a user study to evaluate IABIs and our ratings are subjective assessments solely based on designs of the apps' user interfaces and corresponding logic. Additionally, we discuss our limitations on the lack of a large dataset, automatic testing, and evaluation of the IABI design principles.

User study. The usability problems mentioned in this paper are not verified in a study with end users, resulting in the lack of direct

confirmation of our findings. For example, in T6, we did not test whether an end user is actually misled by the fake lock emoji in the title, even though an expert analysis on the app's user interface strongly suggests the possibility. A possible extension of this work is to design an IABI app practicing GOOD IABI principles and compare end user reactions with those from popular apps tested in this paper.

Ratings. Our evaluations, in particular, the setting of various ratings, are based on previous work [9–11] and World Wide Web (W3C) guidelines on mobile browsers [6]. We believe that some of these settings are subjective assessments and do not advocate the uniqueness of such settings.

Dataset. We conducted tests only on a relatively small dataset with 25 mobile apps. However, these apps are all the most popular ones containing IABIs and are used in daily life. Therefore, we believe that they capture IABIs' representative behaviors experienced by end users. It is worth noting that we do not consider the apps that do not have IABIs, such as *Whatsapp* and *Signal*, because they jump to default browsers when opening a web page.

Towards automatic testing. Our manual testing limits the scalability of this paper. Here we explain briefly why an automatic testing is non-trivial with either dynamic or static approaches.

Unlike existing work on generic dynamic analysis of mobile apps, our analysis of IABI behavior requires triggering *specific* behavior of mobile apps. This typically mandates signing up an account with each app and triggering the processing of specific URLs, which is, to say the least, non-trivial. For example, we cannot find a unified (dynamic) way of precisely locating the (all) chat UI of each app.

Static analysis also encounters specific challenges, e.g., locating the HTTP and HTTPS indicators, which typically do not present themselves as layout files but images. Therefore, it is non-trivial to perform backward tracing to find the relationship between these indicators and the web pages.

Evaluating design principles. The principles in Section 6 present lessons learned from our systematic analysis of the 25 high-profile apps, but they have not been tested in real-world app development or end-product user studies. We hope that our study can bring attention to the community and promote more research on the principles and guidelines for IABIs development.

8 RELATED WORK

In this section, we review some closely related works on the security indicators in regular mobile browsers, the general mobile WebView and TLS security.

Security Indicators in Mobile Browsers. Amrutkar et al. [11] first measured the adequacy of critical security indicators in mobile browsers. They found that mobile browser's UI designs failed to meet many security guidelines. Luo et al. [18] revealed a number of UI vulnerabilities among mobile browsers, which attackers can use to better social engineer users and collect sensitive information. Wu et al. [29] evaluated the usability of mobile browsers' address bars for security guarantees. Different from these studies on standalone mobile browser apps, our study is the first one targeting security indicators in in-app browsing interfaces (IABIs). Moreover, we show that the problems become worse when it comes to IABIs, as many developers only care about the main functionality of the app without putting too much effort into the design of browsing interfaces.

Mobile WebView Security. Android WebView had been vulnerable to various attacks. Luo et al. performed the first study on the attacks of WebView [19, 20], followed by the file-based cross-zone scripting attack [26] and access control problem by Georgiev et al. [14]. Wu and Chang [27] further studied the WebView vulnerabilities on the iOS platform. There are also many techniques to prevent private data from leaking through JavaScript, for example, BavelView [21], Spartan Jester [22], and HybriDroid [15]. Most of the past research focused on the interaction between Java and JavaScript but not on the usability security of the in-app browsing interfaces. For example, Li et al. [17] proposed attacks that utilize browsing interfaces for cross-app navigation from another WebView. Similarly, Yang et al. [30] found that iframe can navigate the WebView to untrusted web pages.

Mobile App TLS Security. Many Android apps use SSL/TLS to transmit sensitive information securely, but developers often use their own, potentially insure, implementation to verify the certificate. Georiev et al. showed that SSL certificate validation is completely broken in various popular apps and libraries [13]. Thus, many previous works studied the potential security threats caused by the inadequate or insecure use of TLS in mobile browsers. They provided various tools to detect potential vulnerabilities against Man-In-The-Middle attacks caused by inadequate use of SSL/TLS, including the dynamic MalloDroid [12] and SMV-Hunter [23] and the static Amandroid [25] and BackDroid [28]. In this paper, we find that most of the apps that use their own IABIs do not have any security indicators about the schemes the website is using, to the extent that the users cannot identify whether the current web page they are browsing meet the TLS requirement.

9 CONCLUSION

In this paper, we conducted the first empirical study on the usability (in)security of in-app browsing interfaces (IABIs) in both Android and iOS apps. Atop a dataset of 25 high-profile mobile apps that contain IABIs, we performed a systematic analysis that comprises eight security tests and covers all the attack surfaces from opening, displaying, to navigating an in-app web page. We obtained three major security findings, including about 30% of the tested apps fail to provide enough URL information before users open the URL, nearly all custom IABIs have various problems in providing sufficient indicators to faithfully display an in-app page to users, and only a few IABIs give specific warnings to remind users of dangerous operations (e.g., password inputting) during navigating a login page. To help mitigate risky IABIs and guide future designs, we reported our findings to affected vendors, analyzed their responses, and proposed a set of secure IABI design principles.

ACKNOWLEDGMENTS

We thank our shepherd, Yasemin Acar, for her comprehensive guidance and the anonymous reviewers for their valuable comments and suggestions. This research/project is partially supported by the Singapore National Research Foundation under the National Satellite of Excellence in Mobile Systems Security and Cloud Security

(NRF2018NCR-NSOE004-0001) and a direct grant (ref. no. 4055127) from The Chinese University of Hong Kong.

REFERENCES

[1] Access in 2021. WebView. http://developer.android.com/reference/android/webkit/WebView.html.

[2] Accessed in 2021. Chrome Custom Tabs. https://developer.chrome.com/docs/multidevice/android/customtabs/.

[3] Accessed in 2021. Chrome Custom Tabs Implementation Guide. https://developer.chrome.com/docs/android/custom-tabs/integration-guide/.

[4] Accessed in 2021. SFSafariViewController. https://developer.apple.com/documentation/safariservices/sfsafariviewcontroller.

[5] Accessed in 2021. UIWebView. https://developer.apple.com/documentation/uikit/uiwebview.

[6] Accessed in 2021. W3C: Web Security Context: User Interface Guidelines. http://www.w3.org/TR/wsc-ui/.

[7] Accessed in 2021. WebViewClient.onPageFinished. https://developer.android.com/reference/android/webkit/WebViewClient#onPageFinished(android.webkit.Web-View,%20java.lang.String).

[8] Accessed in 2021. WebViewClient.onReceivedSslError. https://developer.android.com/reference/android/webkit/WebViewClient#onReceivedSslError(android.webkit.WebView,%20android.webkit.SslErrorHandler,%20android.net.http.SslError)).

[9] Chaitrali Amrutkar, Patrick Traynor, and Paul Oorschot. 2013. An Empirical Evaluation of Security Indicators in Mobile Web Browsers. In *IEEE Trans. on Mobile Computing*.

[10] Chaitrali Amrutkar, Patrick Traynor, and Paul C Van Oorschot. 2012. Measuring SSL indicators on mobile browsers: Extended life, or end of the road?. In *International Conference on Information Security*.

[11] Chaitrali Amrutkar, Patrick Traynor, and Paul C. van Oorschot. 2015. An Empirical Evaluation of Security Indicators in Mobile Web Browsers. *IEEE Transactions on Mobile Computing* (2015).

[12] Sascha Fahl, Marian Harbach, Thomas Muders, Lars Baumgärtner, Bernd Freisleben, and Matthew Smith. 2012. Why Eve and Mallory Love Android: An Analysis of Android SSL (In)Security. In *Proc. ACM CCS*.

[13] Martin Georgiev, Subodh Iyengar, Suman Jana, Rishita Anubhai, Dan Boneh, and Vitaly Shmatikov. 2012. The Most Dangerous Code in the World: Validating SSL Certificates in Non-Browser Software. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*.

[14] Martin Georgiev, Suman Jana, and Vitaly Shmatikov. 2014. Breaking and Fixing Origin-Based Access Control in Hybrid Web/Mobile Application Frameworks. In *Proc. ISOC NDSS*.

[15] Sungho Lee, Julian Dolby, and Sukyoung Ryu. 2016. HybriDroid: Static analysis framework for Android hybrid applications. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*.

[16] Tongxin Li, Xueqiang Wang, Mingming Zha, Kai Chen, XiaoFeng Wang, Luyi Xing, Xiaolong Bai, Nan Zhang, and Xinhui Han. 2017. Unleashing the Walking Dead: Understanding Cross-App Remote Infections on Mobile WebViews. In *Proc. ACM CCS*.

[17] Tongxin Li, Xueqiang Wang, Mingming Zha, Kai Chen, XiaoFeng Wang, Luyi Xing, Xiaolong Bai, Nan Zhang, and Xinhui Han. 2017. Unleashing the Walking Dead: Understanding Cross-App Remote Infections on Mobile WebViews. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*.

[18] Meng Luo, Oleksii Starov, Nima Honarmand, and Nick Nikiforakis. 2017. Hindsight: Understanding the Evolution of UI Vulnerabilities in Mobile Browsers. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*.

[19] Tongbo Luo, Hao Hao, Wenliang Du, Yifei Wang, and Heng Yin. 2011. Attacks on WebView in the Android system. In *Proc. ACM ACSAC*.

[20] Tongbo Luo, Xing Jin, Ajai Ananthanarayanan, and Wenliang Du. 2013. Touchjacking Attacks on Web in Android, iOS, and Windows Phone. In *Foundations and Practice of Security*.

[21] Claudio Rizzo, Lorenzo Cavallaro, and Johannes Kinder. 2018. BabelView: Evaluating the Impact of Code Injection Attacks in Mobile Webviews. In *Research in Attacks, Intrusions, and Defenses*.

[22] Julian Sexton, Andrey Chudnov, and David A. Naumann. 2017. Spartan Jester: End-to-End Information Flow Control for Hybrid Android Applications. In *2017 IEEE Security and Privacy Workshops (SPW)*.

[23] David Sounthiraraj, Justin Sahs, Garrett Greenwood, Zhiqiang Lin, and Latifur Khan. 2014. SMV-Hunter: Large Scale, Automated Detection of SSL/TLS Man-in-the-Middle Vulnerabilities in Android Apps. In *Proc. ISOC NDSS*.

[24] Thomas Steiner. 2018. What is in a Web View? An Analysis of Progressive Web App Features When the Means of Web Access is not a Web Browser. In *Proc. ACM WWW*.

[25] Fengguo Wei, Sankardas Roy, Xinming Ou, and Robby. 2014. Amandroid: A Precise and General Inter-component Data Flow Analysis Framework for Security Vetting of Android Apps. In *Proc. ACM CCS*.

[26] Daoyuan Wu and Rocky K. C. Chang. 2014. Analyzing Android Browser Apps for file:// Vulnerabilities. In *Proc. Springer Information Security Conference (ISC)*.

[27] Daoyuan Wu and Rocky K. C. Chang. 2015. Indirect File Leaks in Mobile Applications. In *Proc. IEEE Mobile Security Technologies (MoST)*.

[28] Daoyuan Wu, Debin Gao, Robert H. Deng, and Rocky K. C. Chang. 2021. When Program Analysis Meets Bytecode Search: Targeted and Efficient Inter-procedural Analysis of Modern Android Apps in BackDroid. In *Proc. IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*.

[29] Min Wu, Robert C. Miller, and Simson L. Garfinkel. 2006. Do security toolbars actually prevent phishing attacks?. In *Proceedings of the SIGCHI conference on Human Factors in computing systems. ACM*.

[30] Guangliang Yang, Jeff Huang, and Guofei Gu. 2019. Iframes/Popups Are Dangerous in Mobile WebView: Studying and Mitigating Differential Context Vulnerabilities. In *28th USENIX Security Symposium*.