Effective Defense Schemes for Phishing Attacks on Mobile Computing Platforms

Longfei Wu, Xiaojiang Du, *Senior Member, IEEE*, and Jie Wu, *Fellow, IEEE*

Abstract—**Recent years have witnessed the increasing threat of phishing attacks on mobile computing platforms. In fact, mobile phishing is particularly dangerous due to the hardware limitations of mobile devices and the habits of mobile users. In this paper, we did a comprehensive study on the security vulnerabilities caused by mobile phishing attacks, including web page phishing attacks, application phishing attacks, and account registry phishing attacks. Existing schemes designed for web phishing attacks on personal computers (PCs) cannot effectively address the various phishing attacks on mobile devices. Hence, we propose MobiFish, which is a novel automated lightweight antiphishing scheme for mobile platforms. MobiFish verifies the validity of web pages, applications, and persistent accounts by comparing the actual identity to the claimed identity. MobiFish has been implemented on a Nexus 4 smartphone running the Android 4.2 operating system. We experimentally evaluate the performance of MobiFish with 100 phishing URLs and corresponding legitimate URLs, as well as phishing apps. The results show that MobiFish is very effective in detecting phishing attacks on mobile phones.**

Index Terms—**Mobile computing, phishing attacks, security and protection.**

I. Introduction

PHISHING attacks aim to steal private information, such as usernames, passwords, and credit card details, by way of impersonating a legitimate entity. Although security researchers have proposed many antiphishing schemes, the threat of phishing attacks is not well mitigated. On the one hand, lots of phishing sites expire and revive rapidly. According to the Anti-Phishing Working Group (APWG), the average time that a phishing site stays online is 4.5 days [1]. Cranor *et al.* even found that it is sometimes on the order of hours [2]. On the other hand, phishing attackers keep improving their techniques so that their new attacks are able to circumvent existing antiphishing tools.

Mobile phishing is an emerging threat targeting mobile users of financial institutions, online shoppers, and social networking companies. In 2012, researchers from Trend Micro found 4000 phishing URLs designed for mobile web pages [3]. Although this number takes up less than 1% of all collected phishing URLs, it highlights that mobile platforms have become new targets of phishing attacks. Notice that mobile users could also be spoofed by conventional phishing web pages (designed for personal computer (PC) browsers) when browsing with their phones. The trend of launching phishing attacks on mobile phones may be attributed to the hardware limitations, such as the small screen size, the inconvenience of user input and application switching, the lack of identity indicators, mobile user habits and preferences, etc.

Almost all phishing attacks on PCs are in the form of bogus websites. Nowadays, with browsers powerful enough to support all kinds of Internet services, people are accustomed to online banking, online shopping, online socializing, etc. They are familiar with being requested to provide, and subsequently providing, private information and credentials to websites. Current phishing detection schemes can be roughly divided into two categories: heuristics-based schemes and blacklist-based schemes. Blacklist-based schemes can detect only those phishing sites that are in the blacklist, and cannot detect zero-day phishing attacks such as those that have only appeared for days or hours. It is possible that new phishing sites may have already stolen user credentials or have expired before being added to the blacklist. Heuristics-based schemes largely depend on features extracted from URL and HTML source code, and then, other techniques, such as machine learning, are used to determine the validity. However, we find that the features extracted from HTML source code could be inaccurate, and phishing sites can easily bypass those heuristics.

Moreover, browsers have many practical features and convenient functions abandoned or truncated during their adaptation to hardware-constrained mobile platforms; this results in an unpleasant experience for users. To improve their services, most well-known enterprises have published mobile applications (apps) for major mobile platforms. This sheds new light on phishing scams: some phishing attackers develop fake apps, or repackage legitimate apps, and then upload these phishing applications to unofficial app markets. Once the attack succeeds, the victim's credentials will be sent to the phishing server. Phishing apps are even harder to detect than phishing web pages, since for web pages, we are able to judge the destination of form data from HTML source code (*action* attribute in the *form* tag). However, for mobile apps, there is no way to check if user credentials are sent to the legitimate authentication server or the attacker's server. Hence, phishing attacks on mobile phones are more complicated than those on PCs. It is important to design effective mobile phishing defense schemes for both web pages and applications.

In addition, we further discover a specialized form of phishing attacks, which target the persistent account registry

Manuscript received March 13, 2015; revised June 30, 2015; accepted August 18, 2015. Date of publication August 25, 2015; date of current version August 11, 2016. This work was supported in part by the U.S. National Science Foundation under Grant CNS-0963578, Grant CNS-1022552, and Grant CNS-1065444. The review of this paper was coordinated by Prof. W. Song.

The authors are with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122 USA (e-mail: longfei.wu@temple.edu; dux@temple.edu; jiewu@temple.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVT.2015.2472993

function of mobile operating systems (OSs). Since the malicious apps that have created a persistent account interact with the user using a separate login interface (not the app's login interface), we also need to design a defense scheme that is specific to account registry phishing attacks.

In this paper, we propose a novel lightweight antiphishing scheme for mobile devices, i.e., MobiFish, which is capable of defending against phishing attacks on mobile web pages, apps, and persistent accounts. MobiFish aims to solve the essential problem of identity masquerade, without reliance on HTML source code, search engine, or machine learning techniques. We employ the optical character recognition (OCR) technique to extract text from the screenshot of a login interface, which achieves better performance on mobile phones than on PCs. We are able to find the claimed identity, from the extracted text, and the actual identity, from the URL of a web page or remote server (for mobile apps). If these two identities are different, our tool issues a warning to the user. Regarding the account registry phishing attacks, MobiFish can effectively detect all three variants by checking the consistency of account label, app name, and the destination of credential transmission.

Our contributions are summarized as follows:

- We find the weakness of previous heuristics-based security schemes for conventional web page phishing, and we propose a lightweight detecting strategy that utilizes OCR for web phishing and app phishing attacks.
- We present account registry phishing attacks. To the best of our knowledge, we are the first to give detailed formulation and defense scheme for these types of attacks.
- We propose MobiFish, a novel automated lightweight antiphishing scheme for mobile phones. We implement MobiFish on a Google Nexus 4 smartphone (Android 4.2).
- We evaluate the effectiveness and usability of MobiFish with phishing URLs and phishing apps. We also measure the delay overhead of MobiFish.

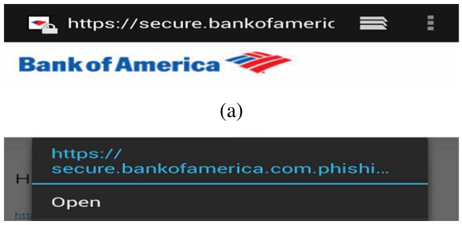
The remainder of this paper is organized as follows: Section II summarizes the factors making phone users susceptible to mobile phishing attacks. Section III presents the mobile phishing attack models. Section IV provides an overview of the MobiFish scheme. Section V describes the detail of the MobiFish scheme. Section VI shows our evaluation methodology and results. Section VII discusses the related works. Section VIII concludes this paper.

II. Why Users are Susceptible to Mobile Phishing

Here, we conduct a comprehensive analysis on the factors that make mobile users vulnerable to phishing attacks, from the objective perspective of hardware limitations and the subjective perspective of mobile users themselves, respectively.

A. Hardware Limitations

Due to the small size of phone screens and limited computational power, browsers in mobile systems have to remove or degrade some features to make more space for web contents and maintain a smooth user experience (e.g., loading speed). However, the security-related functionalities are among those missing features. As a result, phishing web pages that could have been detected and blocked on PC browsers may still be accessible from mobile browsers.


https://secure.bankofameric

(a)

https://
secure.bankofamerica.com.phishi...

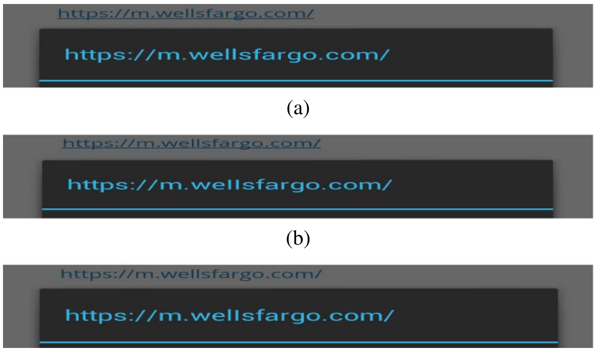
(b)

Fig. 1. Display of URL in mobile browser (Android). (a) Display of partial URL. (b) Display of link destination.

In addition, the user interface (UI) of mobile browsers is also simplified, which could instead help phishing sites to bypass user inspection. To accommodate the small phone screen size, most mobile browsers have to remove the status bar and hide the URL bar once the web pages finish loading. Even during the loading process, long URLs are truncated to fit the browser frame. Since the ability to read and verify URLs is crucial in detecting phishing attacks, partial URL (in particular, a URL with only partial domain name displayed) would certainly increase the user's risk of being spoofed by the phishing attacks. For example, Fig. 1(a) shows the URL bar with only a partial domain name when loading the "Bank of America" website. This could lead to a successful phishing attack if users are convinced by the partial URL and submit their credentials, while the full URL turns out to be "https://secure.bankofameric.com" or "https://secure.bankofamerica.com.phishing.com." Such tricks would fail if the entire URL (or at least the full domain name) is displayed. One possible way by which a user can view the complete URL is to click the address bar and manually scroll all the way to the end. Another way is to view the actual destination of a link, which can be invoked by holding the link for about 2 s. Although the destination URL is also partially presented as in Fig. 1(b), it can display the domain name with as many as 31 characters, instead of 19 characters in the URL bar. Since the full domain names of the login pages are no longer than 30 characters for most legitimate sites, checking the destination allows users to detect phishing sites more quickly.

Moreover, for some legitimate sites, their domain names could be easily mimicked by replacing the letters. For example, it is hard to distinguish "l" from the capital "i" because mobile browsers display them both in vertical slash shape [e.g., Fig. 2(a) and (b)]. In Fig. 2(c), we list both "l" and capital "i" together and find that their small difference in height is difficult to discern with human eyes. For this kind of letter replacement phishing attacks, even attentive and observant users who always check the entire URL (domain name) are likely to be fooled.

In addition, the lack of identity indicator is another issue in mobile phishing. Unlike the URL bar in browsers, there is no straightforward indication of identity available for mobile apps and persistent accounts. Users may manually look up the task


https://m.wellsfargo.com/
https://m.wellsfargo.com/
https://m.wellsfargo.com/
https://m.wellsfargo.com/
https://m.wellsfargo.com/
https://m.wellsfargo.com/

(a)

(b)

(c)

Fig. 2. URL letter replacement. (a) Wells Fargo domain name. (b) Wells Fargo domain name with "l" replaced by capital "i." (c) Wells Fargo domain name with "l" and capital "i."

list to check the identity of the running apps. However, it is not possible to verify to which app a persistent account has been bound.

B. *User Habits and Preferences*

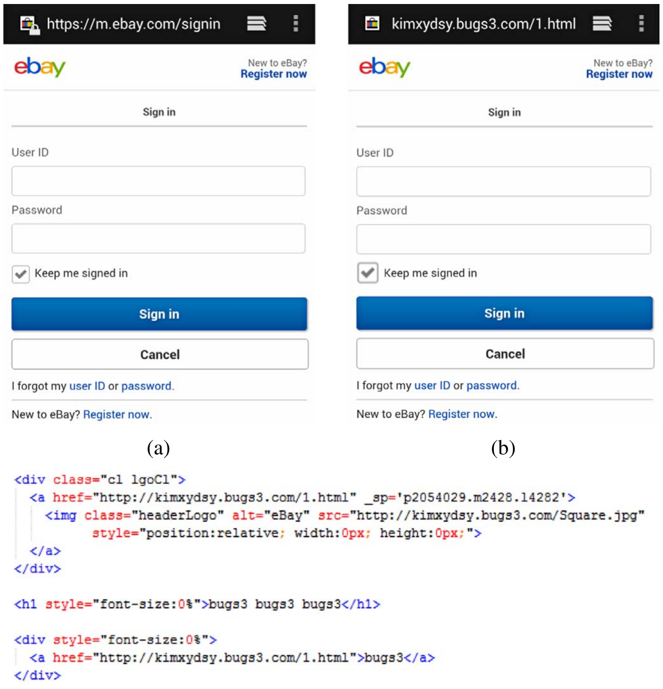
The habits and preferences of mobile users further increase the vulnerability to mobile phishing attacks. Over the past few years, touchscreen smartphones have become dominant in the mobile phone market. However, typing on a virtual keyboard is not as easy as on a physical keyboard due to the lower input accuracy, particularly when walking or sitting in a moving vehicle. Because of this, it is tempting to follow links in web pages or e-mails, rather than typing the links manually. Another factor is that on smartphones, switching among applications or even shifting to other pages within a browser is more complicated and tedious than being performed on a PC. Users who value convenience usually prefer to follow links from other applications [4]. In addition, phishing attacks can succeed because users have become accustomed to entering their credentials in familiar and repeated login interfaces. If users frequently encounter legitimate links whose targets prompt them for private data, then they get used to reflexively supplying the requested data [5].

III. Mobile Phishing Attack Models

Here, we present the three types of phishing attack models studied in this paper.

A. *Mobile Web Page Phishing Attacks*

Web page phishing detection has been widely studied and applied in PC browsers. Blacklist-based matching and heuristics-based detection are the two major existing methods used for web page phishing detection. The blacklist method is to search a suspicious site in a list of reported phishing sites. Although it can achieve high accuracy at the cost of human verification, the delay in updating the blacklist would greatly degrade its effectiveness. Specifically, blacklist-based methods cannot defend new phishing sites that have not been listed, such as zero-day phishing attacks. Heuristic detection methods are based on features extracted from URL and HTML source code, and they often work with the assistance of search engine or machine learning techniques. These features are summarized from previously reported phishing sites. However, a new phishing site may not have these features at all because each feature only appears in some of the phishing samples. This means carefully constructed phishing sites that remove all suspicious features are able to bypass the heuristics-based detections (this is why heuristics-based approaches cannot achieve a 100% detection rate).


https://m.ebay.com/signin
ebay
New to eBay?
Register now
Sign in
User ID
Password
☑ Keep me signed in
Sign in
Cancel
I forgot my user ID or password.
New to eBay? Register now.
kimxydsy.bugs3.com/1.html
ebay
New to eBay?
Register now
Sign in
User ID
Password
☑ Keep me signed in
Sign in
Cancel
I forgot my user ID or password.
New to eBay? Register now.

(a)

(b)

```
<div class="cl lgoCl">
  <a href="http://kimxydsy.bugs3.com/1.html" _sp='p2054029.m2428.14282'>
    <img class="headerLogo" alt="eBay" src="http://kimxydsy.bugs3.com/Square.jpg"
          style="position:relative; width:0px; height:0px;">
  </a>
</div>

<h1 style="font-size:0%">bugs3 bugs3 bugs3</h1>

<div style="font-size:0%">
  <a href="http://kimxydsy.bugs3.com/1.html">bugs3</a>
</div>
```

(c)

Fig. 3. Comparison of real and fake eBay login pages. (a) eBay official login page. (b) eBay phishing login page. (c) Inserted HTML source code.

In addition, we find that information extracted from HTML source code may not be able to reflect the web page displayed to the user. This is because attackers can add text, images, and links into HTML source code; meanwhile, they can also make "undesirable" content disappear from a web page by simply changing its size or covering it with other images. Therefore, features like word frequency, brand name, and company logo could be easily manipulated. For example, Fig. 3(a) shows the real eBay mobile login page. We copy the code of the original site and migrate it to our website. We also upload the image components to our website and change the links to the corresponding places within our website, particularly its form-data submission URL. Then, the code segment in Fig. 3(c) is added into the source code. However, the tampered web page [see Fig. 3(b)] looks exactly the same as the official eBay site. No user will suspect its validity without looking at its URL. Meanwhile, phishers can insert as many "bugs3" as needed into the HTML source code to obfuscate conventional identity extractors. The large number of "bugs3" extracted is able to convince the identity extractor that this web page claims to be "bugs3" instead of "ebay." As a result, antiphishing heuristics would fail since the phishing web page indeed belongs to the "bugs3.com" domain. The title of a web page is not visible unless the user clicks the page menu icon and switches to an overview of the opened page list, which means the title could also be replaced by "bugs3" to enhance the consistency of HTML source code. Hence, HTML source code is not a reliable source for phishing web page detection. We seek a new approach to extracting the identity of a web page.

B. Mobile Application Phishing Attacks

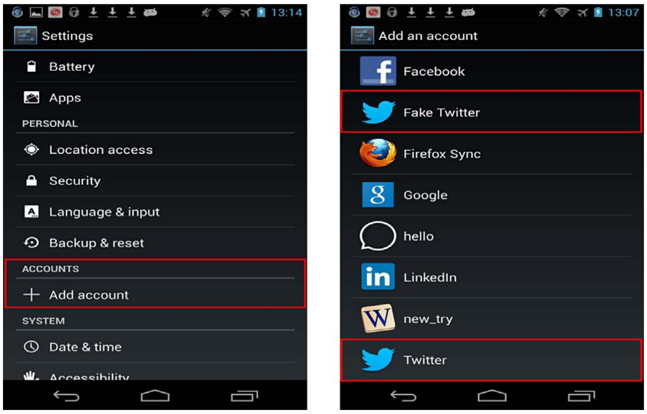
Phishing attacks based on applications are quite uncommon in PCs, but they are disturbing problems on mobile platforms. According to Felt and Wagner [6], the four high-risk phishing attack models with a "common" prevalence level and a "perfect" accuracy level are all associated with mobile applications that impersonate legitimate apps. Application-oriented phishing attacks can be further categorized into two types based on the way in which they are launched: some phishing applications attempt to hijack existing legitimate apps. These phishing apps continuously perform task polling, and they launch themselves as long as they detect the launch of the target apps. As a result, the fake login interface layers over the top of the real one, and the phishing app "appears" to be the target app. Mobile users do not know what has happened since everything is accomplished during a single window switching process. One possible way to solve this is to check the identity of the current foreground app from the task list, although, normally, no user does that. Another type of phishing app directly shows up as the target app. This may occur when a user downloads fake apps from unofficial app markets.

Despite the various methods of stealing user credentials, the essential attack pattern must end with the transmission of credentials to the attacker. Hence, runtime monitoring and blocking the communication of the suspicious apps can effectively defeat the app phishing attacks.

C. Mobile Account Registry Phishing Attacks

The Android system provides a centralized account registry and management function, which allows the phone users to log in to their online accounts (e.g., Google, Facebook, Twitter, etc.) in a once-for-all manner; we call this function "Persistent Account Registry." It is very convenient in that phone users will not be bothered to enter credentials each time they launch a social networking app or a financial institution app. Since most phones are in personal use and serve only the phone owner, the user's privacy can be guaranteed by simply setting an "overall" screen locking password. In the Android system, the persistent account registry can be accessed through "Add account" in the "Settings" directory, as depicted in Fig. 4(a).

Fig. 4(b) shows a list of applications that support persistent account registry. Basically, any app can register itself to show up in the account list as a new account type. The app should have an authentication service, which is bound to the *AccountManagerService* of the Android system. This service must specify the following intent filter (i.e., with action ACTION_AUTHENTICATOR_INTENT) and corresponding metadata tags in the manifest file.



(a) (b)

Fig. 4. Persistent account registry function. (a) Settings directory. (b) Account list.

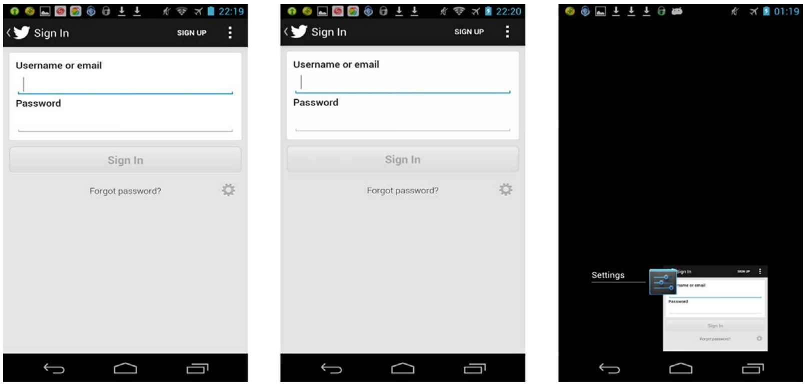
```
<intent-filter>
  <action
    android:name="android.accounts.AccountAuthenticator" />
</intent-filter>
<meta-data
  android:name="android.accounts.AccountAuthenticator"
  android:resource="@xml/authenticator" />
```

To make an app appear properly in the account list, the *android:resource* attribute in the metadata will point to an *xml* resource file that contains at least the following three attributes: *android:accountType* is the name of the new account type, which uniquely identifies this account/app; *android:icon* and *android:label* are the icon and label displayed in the account list, respectively. In addition, *android:smallIcon* is used in the contact application's tab panel, and *android:accountPreferences* points to an *xml* hierarchy, which contains the *PreferenceScreens* that can be invoked.

```
<account-authenticator
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:accountType="typeOfAuthenticator"
  android:icon="@drawable/icon"
  android:smallIcon="@drawable/miniIcon"
  android:label="@string/label"
  android:accountPreferences="@xml/account_preferences" />
```

When a certain account type is selected to be added by the phone user, the authentication service of the corresponding app will be invoked and it will create an *authenticator* instance. The *authenticator* class extends the *AbstractAccountAuthenticator* class, overrides the *addAccount* method, and will launch the login interface in which the user is prompted to enter the credential.

We found that persistent account registry is vulnerable to phishing attacks as well. The icon and label displayed in the account list are defined by corresponding applications, and they are not necessarily the same as those used in the main menu. In other words, any third-party app could register itself as another entity in the account list. For example, a malicious game app may pretend to be an app of a social networking company or financial institution.



(a) (b) (c)

Fig. 5. Real and fake Twitter login UIs. (a) Real Twitter UI. (b) Fake Twitter UI. (c) Task list.

Our demo app shows up as a "Fake Twitter" app in the account list [as in Fig. 4(b)], and its login interface [see Fig. 5(b)] looks exactly the same as the legitimate "Twitter" app [see Fig. 5(a)]. The difference is that, when a user clicks the "Sign In" button, the credentials will be sent to the attacker, instead of the Twitter authentication server. In the app phishing attacks, we have dealt with an indistinguishable fake login interface using the AppFish scheme. However, the account phishing attack is more stealthy. When the login interface is present to the user, the user does not know which app it truly belongs to. As in Fig. 5(c), even if a phone user checks the task list, *Settings* is the only app in there. The untraceable nature of the account registry interface leaves it out of the protection of AppFish.

In the design of an account phishing app, a practical issue has to be solved: If there appears to be two "Twitter" apps in the account list, the user could immediately know the abnormality. To avoid duplicates, the malicious app should perform a scan of all installed apps before registering a phishing account. In fact, the phishing app could incorporate multiple sets of authenticators (including the authentication service, login interface, etc.), each for one specific account type. This means that the app has the potential to masquerade as multiple legitimate apps. Each time, it only chooses to be one of the apps that has not been installed on the victim device. The polling of the installed apps can be easily accomplished using *PackageManager*, and it is stealthy in that no permission is required. However, if all the target apps have been installed, then no account phishing attack will be launched.

It may sound impossible to load the *xml* resource file (where the account icon and label are defined) dynamically, since it is specified explicitly in the manifest file and will only be read once (during the installation process). However, we have discovered a tricky bypass to this limitation. As the *xml* resource file is referenced in the metadata tag of the corresponding authentication service, the account information contained in the *xml* file will not be registered if that service is "disabled." Initially, we set the *android:enabled* attribute of all authentication services to *false*. As soon as the account phishing app is installed, it scans the installed apps and decides which account to impersonate. Then, it calls the *setComponentEnabledSetting* method of *PackageManager*, which can override the "disabled" state that has been set by the service in the manifest file. For example, if the legitimate Twitter app is missing on the phone, the phishing "Twitter" authentication service will be enabled, and the fake "Twitter" account will be registered in the account list during runtime. Furthermore, the account phishing app needs to periodically check the installed app list. If duplicates are detected (user later installed the legitimate app), it could switch to another account type/app that has not been registered/installed.

Note that although Android uses several permissions to regulate persistent accounts (e.g., ACCOUNT_MANAGER, AUTHENTICATE_ACCOUNTS, MANAGE_ACCOUNTS, USE_CREDENTIALS, and GET_ACCOUNTS), they are either specified for the authentication process, i.e., the retrieving of the existing accounts information, or are reserved only for system apps. Our demo app "Fake Twitter" [as shown in Figs. 4(b) and 5(b)] can appear in the account list, present the fake login interface, and request user credentials without any account-related permission.

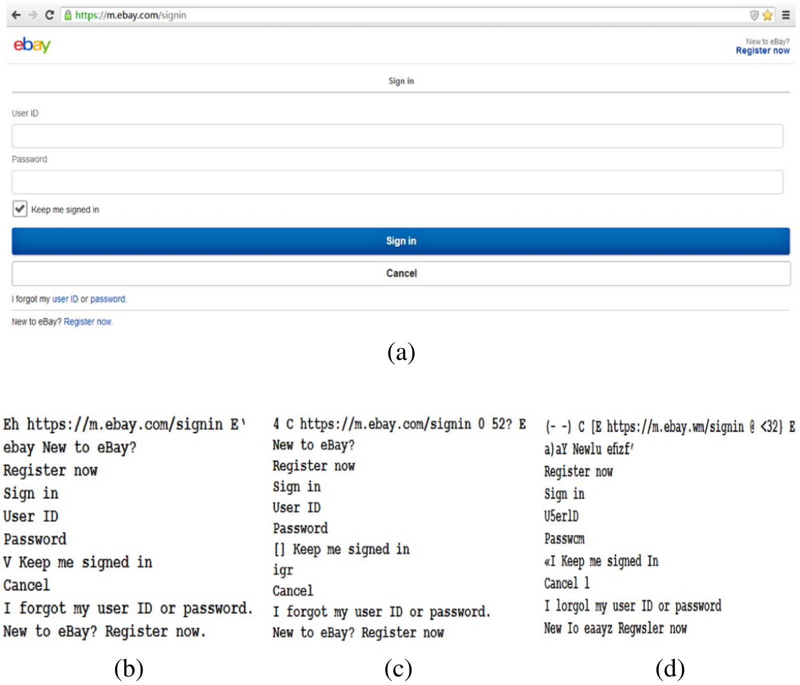
IV. Overview of MobiFish Scheme

A. Motivation

Phishing attackers take fancy tactics to direct victims to their phishing sites or applications, which masquerade as trustworthy entities. The key of solving the phishing problem is to find the discrepancy between the identity it claims to be and the identity it actually is. We have shown that HTML source code is not a reliable clue to find the claimed identity of a phishing site. As an alternative, we should focus on the screen presented to the user since users are directly spoofed by what they see. In addition, existing antiphishing schemes cannot detect app phishing attacks and account registry phishing attacks. Thus, there is a strong need for an effective defense scheme against the phishing attacks on mobile platforms.

B. Identity Extraction

As discussed earlier, the screen presented to mobile users should be the exact place where the claimed identity is extracted from. It turns out that a good way to capture the screen content is to take a screenshot. There are two common observations that lead us to believe that screenshots can work well in identity


https://m.ebay.com/signin
ebay
New to eBay?
Register now
Sign in
User ID
Password
☑ Keep me signed in
Sign in
Cancel
I forgot my user ID or password.
New to eBay? Register now.

(a)

```
Eh https://m.ebay.com/signin E'
ebay New to eBay?
Register now
Sign in
User ID
Password
V Keep me signed in
Cancel
I forgot my user ID or password.
New to eBay? Register now.
```

(b)

```
4 C https://m.ebay.com/signin 0 52? E
New to eBay?
Register now
Sign in
User ID
Password
[] Keep me signed in
igr
Cancel
I forgot my user ID or password.
New to eBay? Register now
```

(c)

```
(- -) C [E https://m.ebay.wm/signin @ <32} E
a)aY Newlu efizf'
Register now
Sign in
U5erlD
Passwcm
«I Keep me signed In
Cancel l
I lorgol my user ID or password
New Io eaayz Regwsler now
```

(d)

Fig. 6. Comparison of OCR performance on mobile phone and PC. (a) eBay official login page in PC browser. (b) Text extracted from Fig. 5(a) with Tesseract. (c) Text extracted from Fig. 6(a) with MODI. (d) Text extracted from Fig. 6(a) with Tesseract.

extraction and verification. The first is that most login interfaces of legitimate mobile sites and apps are very simple. The entire login page, or the majority of the page, can be captured in one screenshot. Another observation is that the brand names and the company logo (identity) locate at apparent places in the login page, which can be easily captured and extracted from the screenshot. Screenshots can be used for the phishing detection in both web pages and applications. Since the source code of apps are not available, there is no way to acquire the content of an app login interface other than taking a screenshot. Then, to obtain the claimed identity, the OCR technique is utilized to convert the screenshot into text.

The actual identity of a mobile web page can be obtained from the second-level domain name (SLD). Most well-known enterprises use their brand names as the SLD of their official websites. This can be best illustrated by Bank of America (BoA). BoA uses the entire brand name as the SLD, despite its length. In special cases that brand names are not exactly the same as the SLD, e.g., "AT&T," which contains a symbol in the brand name, all content-based schemes will fail due to the mismatch of brand name "AT&T" and SLD "att" (for legitimate URLs, special symbols are usually not included in domain name). However, such inconsistencies can be easily solved with a mapping whitelist (MWL), in which the brand name "AT&T" is mapped to the SLD "att" before the identity verification, or *vice versa*. Again, due to the unknown source code, the actual identity of mobile apps cannot be decided until the transmission of user credentials happens. The checking of the suspicious apps must be cleared before they are allowed to transmit.

In addition, to detect malicious apps launching account phishing attacks, we first need to ensure that the identity shown in the main menu (app name) and the identity displayed in the account list (account label) are consistent. However, even if this condition is satisfied, it could still be a repackaged legitimate app. We need to find out its actual identity by tracking where the credentials are sent to, as the same in the AppFish scheme.

C. OCR Techniques

OCR is the mechanical or electronic conversion of an image to machine-encoded text. According to previous works, it is valuable for phishing detection as long as high-quality OCR solution is used [7]. OCR has been utilized to extract text from simple text-only logo in [8]. GoldPhish [9] uses OCR to read text from a web page (specifically the company logo). We believe that the OCR technique could achieve better performance on mobile phones because phones have a smaller screen size and a relatively higher pixel density.

We deploy the OCR technique into the mobile platform and show that it achieves better performance and effectiveness on mobile phones by real experiments. The tool that we use is Tesseract [10], which is one of the most accurate open source OCR engines, and supports over 60 languages. Our testing uses a ThinkPad T420 laptop (2.40 GHz, 4-GB RAM) with a pixel density of 131 dpi and a Google Nexus 4 smartphone (1.5 GHz, 2-GB RAM) with 320 dpi.

We open the eBay mobile login page in both mobile and PC browsers, and each captures a screenshot [as shown in Figs. 3(a) and 6(a)]. Tesseract is used to extract text from the screenshot taken on the mobile phone, while Microsoft Office Document Imaging (MODI) is used for the screenshot of PC browser (this tool is used in GoldPhish [9]). The results are given in Fig. 6(b) and (c), respectively. We find that Tesseract extracts all words correctly from the mobile screenshot, except the "Sign in" in the dark blue button. The performance of MODI on PC is not as good as Tesseract, because MODI not only missed the dark blue button but missed the word “ebay” in the top-left logo as well. This example also shows the ability of Tesseract to deal with various styles and fonts of text in company logos. Moreover, the OCR extraction on mobile phones only takes 1.6 s, while on PC, the time is 4.5 s. To mitigate the influence of different OCR engines, we also extract the screenshot of a PC web page using Tesseract (Windows version). Although it takes only 1.5 s, the accuracy is much worse because as many as ten words are extracted incorrectly, including the eBay logo [see Fig. 6(d)].

The aforementioned tests show that OCR achieves higher accuracy and efficiency on the mobile platform. It plays an important role in the identity extraction module of our mobile antiphishing scheme.

V. MobiFish Scheme

Here, we present an automated lightweight scheme for mobile phishing defense named MobiFish. MobiFish consists of three major components named WebFish, AppFish, and AccountFish, which are designed to protect mobile web pages, applications, and persistent accounts, respectively.

A. WebFish Scheme

The work flow of WebFish is given in Fig. 7. As we can see, the defense scheme is initiated with URL loading. When a browser attempts to load a web page, WebFish first scans its URL to see whether the domain name is an IP address. Legitimate websites always use domain names as a verification of their identities, while phishers are likely to use an IP address to disguise their fake identities. Next, WebFish obtains the HTML source code of the loading page and checks if there is any form in that page. Like legitimate login pages, phishing web pages also need a form with an *input* tag, which allows the user to enter (confidential) information and then submit. WebFish checks the existence of forms so that not every page has to go through the checking. However, the core module of identity extraction does not rely on any part of HTML source code. If a form is found, WebFish starts the identity extraction and verification. On one hand, it extracts the SLD from the URL, which represents the actual identity of the site. Then, the SLD is indexed in the MWL. If it matches any of the SLD–brand name records in the MWL, the original SLD is replaced by the corresponding brand name. On the other hand, it calls the *screencap* native function to take a screenshot of the login page and extract the text with the OCR tool. Note that the URL shown in the URL bar may also be captured into the screenshot, and it should be removed from text before identity verification, as it contains the actual identity (SLD) of the site. The existence of a URL bar in the screenshot can be determined by whether the first line contains one of the top-level domain names (e.g., gov, edu, com, and org). To further speed up the detection process, we search for sensitive terms, such as “username,” “password,” and “credit card number,” in the text. If not found, the form may be just used for search or general data input purposes, and the page is marked as safe directly. Otherwise, the last step is to search the SLD in the text. If not found, it is marked as a phishing site. WebFish shows a notification window to the user, indicating the high possibility of a phishing attack, along with the URL of the suspected web page.

Our design is based on the assumption that, if the domain name of the phishing site appears in the fake login page of a legitimate entity, the user can immediately discern the difference and check the URL to verify the validity of this web page. This is reasonable since, as far as we know, no phishing site uses common terms in login pages like “sign,” “username,” “password,” or “welcome” as the SLD. Legitimate mobile login pages are made very simple and clear. It is highly unlikely for these well-constructed and well-maintained web pages to have strange words (SLD of phishing sites) appear in them. Thus, users would become alerted if a web page contains text different from the brand name or common login terms. If the attacker adds the phishing domain name in tiny font size to prevent the user from noticing, then the OCR is not able to recognize it either, and WebFish will still mark it as a phishing site. The key feature for WebFish to detect a phishing URL is that the SLD is not among the text extracted from the screenshot of the login page.

B. AppFish Scheme

The work flow of AppFish is shown in Fig. 8. AppFish maintains a database called Suspicious App Set (SAS), which contains the profiles of untrusted apps, including the user ID (Uid), the launching time, and the screenshot text. Users can add the apps that they suspect into SAS, and only apps listed in SAS are under the monitoring of AppFish. These apps can be characterized as follows.

1) Specified for One Company: This is to ensure that the app only contacts the company’s official or affiliated (partners) servers. The domain names of the collaborators are collected and added into the SAS profile in advance. Owning multiple domains often happens to websites that need extra storage. For example, we find in our testing that Facebook may request data from domains like fbcdn.net and akamaihd.net. This is because Facebook uses them as a content delivery network (CDN). The substantial amount of photos generated by Facebook users are uploaded to akamaihd.net, instead of facebook.com. Whenever a user wants to view a photo, the request is actually sent to the nearest *akamaihd* server.

2) Require User Sign In: There are lots of apps that do not need users to log in, like browsers and apps for news, music, maps, etc. In these cases, phishing attacks would not happen at all. For browsers, web page login is protected by WebFish.

The AppFish defense scheme works in two phases: launching phase and authentication phase. In the launching phase, AppFish obtains the name of each launching application and searches for it in SAS. If found, the logging process begins, in which AppFish takes a screenshot of the login interface and extracts the text with the OCR tool. Then, the text along with the application Uid and the launching time are logged into the profile of that app. After the user has entered the credentials and clicks the “Sign in” button, the authentication phase begins. Legitimate applications (like Facebook and Twitter) usually send the user’s credentials to a remote server for authentication via HttpGet/HttpPost. Once the credentials are

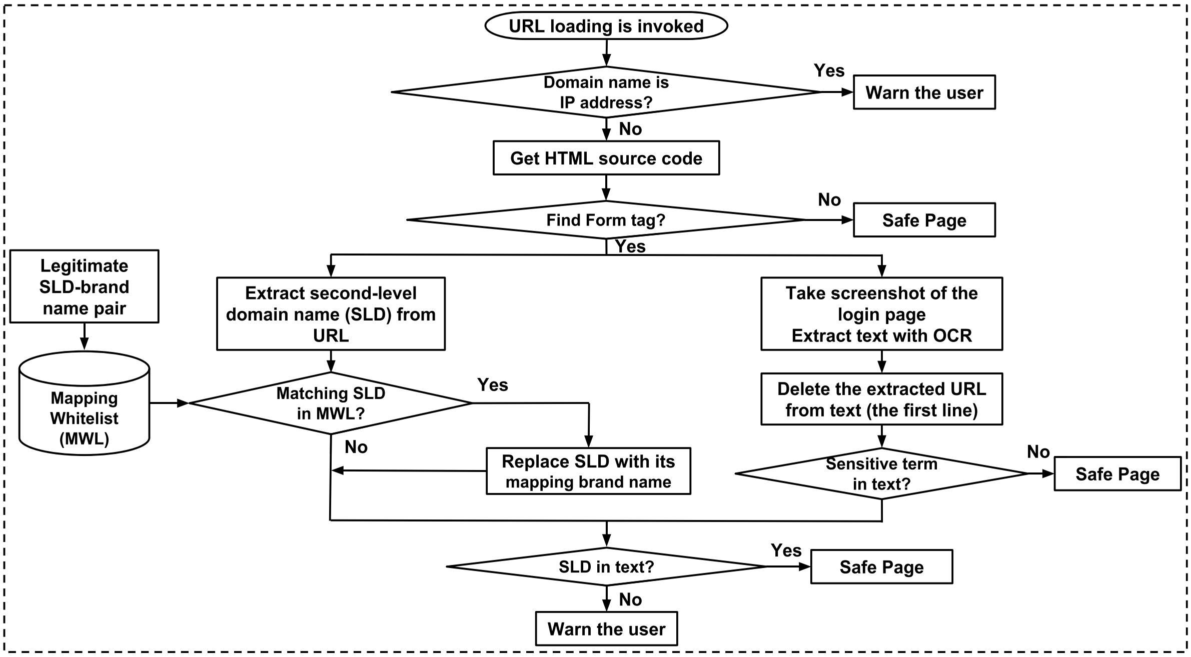

URL loading is invoked
Domain name is IP address?
Yes
Warn the user
No
Get HTML source code
Find Form tag?
No
Safe Page
Yes
Legitimate SLD-brand name pair
Extract second-level domain name (SLD) from URL
Take screenshot of the login page Extract text with OCR
Mapping Whitelist (MWL)
Matching SLD in MWL?
Yes
No
Delete the extracted URL from text (the first line)
Replace SLD with its mapping brand name
Sensitive term in text?
No
Safe Page
SLD in text?
Yes
Safe Page
No
Warn the user

Fig. 7. Work flow of WebFish.

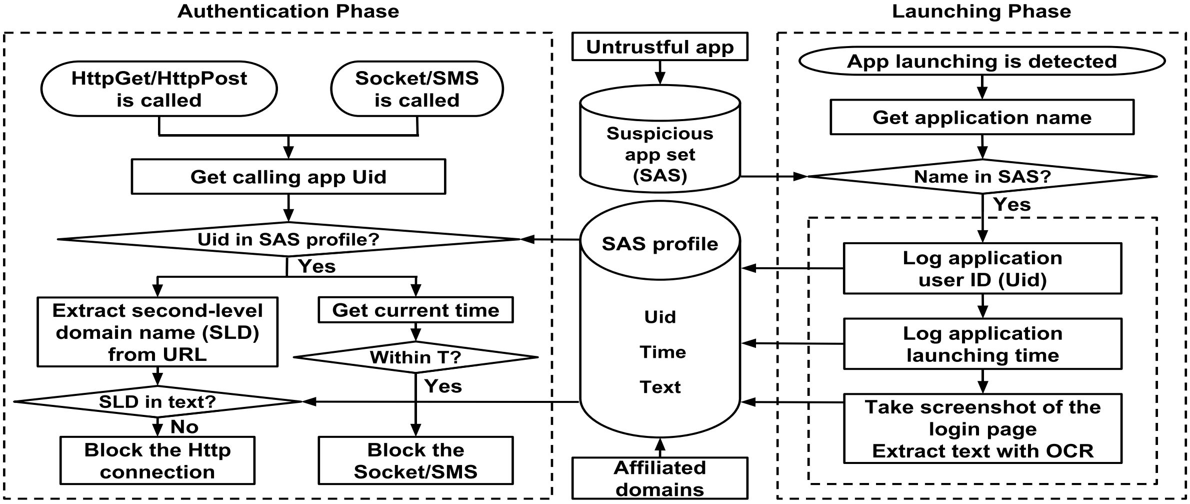


Fig. 8. Work flow of AppFish.

verified, the application loads data belonging to that account. On the contrary, phishing apps are not able to load user data, and the only trick they can play is to tell the user that he or she has entered the wrong password. However, after two or three times, most users will suspect the validity of the application and will uninstall it. Hence, a phishing app is able to send out the user credentials only during the period from submission to uninstallation. AppFish monitors the possible paths for a phishing app to transmit data to the outside world, which include HttpGet/HttpPost, socket, short message services (SMS), and e-mail (e-mail is also based on socket). If an application uses any of these means to send out information, AppFish checks whether it is one of the suspects in SAS. If confirmed, http connections (HttpGet/HttpPost) are filtered, while other communications (socket/SMS) are blocked. For all URLs with which the suspicious app requests to connect, AppFish ensures that the SLD name appears in the screenshot text or the affiliated domain names stored in the SAS profile. Meanwhile, socket and SMS functions are blocked for a certain amount of time T, which should be long enough for the user to notice the abnormality and uninstall the malicious app. Thus, for phishing applications, they will not be able to send out credentials before being removed by the user.

Note that we have to search the SLD within the text extracted from the login page. The reason why extracted text (instead of the application name) is used is that for phishing attacks based on task interception (hijacking), the phishing app can have the same app name as the SLD of the phishing server, while it can pop up a fake login page in another entity. For instance, a mobile user downloads a phishing app named “abc,” due to its tempting fancy functions. However, this phishing app could pop up a fake Facebook login interface as soon as the legitimate

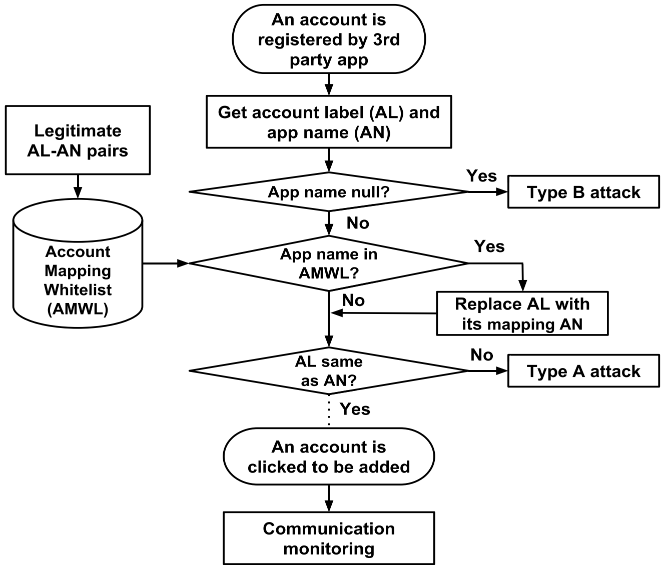

An account is registered by 3rd party app
Get account label (AL) and app name (AN)
Legitimate AL-AN pairs
App name null?
Yes
Type B attack
No
Account Mapping Whitelist (AMWL)
App name in AMWL?
Yes
No
Replace AL with its mapping AN
AL same as AN?
No
Type A attack
Yes
An account is clicked to be added
Communication monitoring

Fig. 9. Work flow of AccountFish.

Facebook launches. Once the user is spoofed, the app "abc" immediately sends the credentials to the phishing server "abc.com." In this example, the foreground (fake) application name "abc" is the same as the SLD of the phishing server, but it cannot be found in the "Facebook" phishing login interface.

C. AccountFish Scheme

The unlimited registry of persistent accounts among third-party apps poses a huge threat to mobile user privacy and account security. According to Yahoo Aviate's collected data in 2014 [11], the average number of apps installed for an Android user is 95. Users may not remember clearly what app has been installed, and hence, they are vulnerable to account registry phishing attacks. We propose AccountFish to defend against the phishing attack targeted at persistent accounts. The work flow of AccountFish is described in Fig. 9.

The account registry phishing attacks can be classified into three types, based on the identities that the malicious app appears to be in the main menu and the account list. In the type-A attack, the malicious app appears to be a different app to the target account (e.g., a game app registers a Twitter account). In the type-B attack, the malicious app does not appear in the main menu at all. In the type-C attack, the malicious app directly shows up as the target app (e.g., repackaged app), which means that they will have the same application name shown in the main menu as the account name that appears in the account list.

The idea of the detection mechanism for the type-A and type-B account phishing attacks is to compare the app name in the main menu and the account label in the account list. As mentioned earlier, the malicious app can dynamically register and change the account information. AccountFish should be able to inspect the registration of accounts in runtime, which can only be accomplished by modifying the Android source code. Specifically, we modified the *parseServiceAttributes* method of *AccountAuthenticatorCache* class so that the account label is extracted and compared with the host app name each time an account is being added (before the corresponding authentication service is called). If the app name and the account label are inconsistent, then that app is highly likely to be a malicious app. However, there are a few legitimate apps whose account labels are not exactly the same as the corresponding app names [e.g., "Firefox Sync" in Fig. 4(b)]; we solve this issue using an account MWL (AMWL) that contains all the "inconsistent" legitimate apps. For suspicious apps that are not in the whitelist, a warning containing the app and account names will be issued instantly. The apps in AMWL and those having consistent names will go through further checks (type-C attack detection).

The mechanism that we use to detect the type-C attack is similar to AppFish, because the decision cannot be made until the transmission of credentials happens. The difference is that a screenshot is unnecessary in AccountFish: The "add account" event is not available to other third-party apps; hence, the account login interface cannot be hijacked. We add a hook in the *bindToAuthenticator* method of *AccountManagerService*, which can catch the user click event for adding a specific account and launch the communication monitoring mechanism. Specifically, the app name (account label) will be directly used to filter the outgoing Http connections (HttpGet/HttpPost); only URLs with the same SLD as the app name are permitted. Meanwhile, other communication channels of the suspected app (socket/SMS) are blocked for a certain amount of time T, which is long enough for a user to notice the abnormality and uninstall the malicious app.

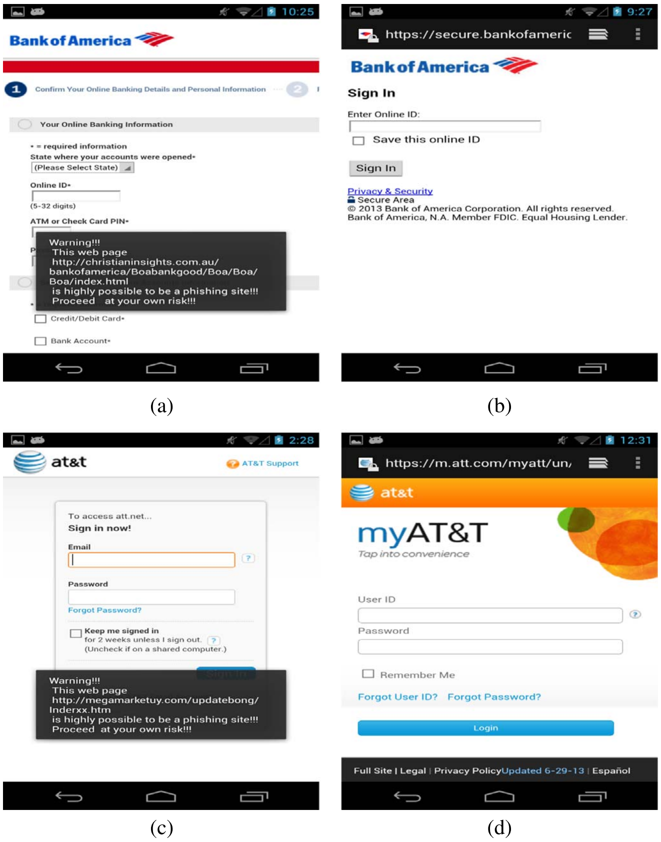
VI. Implementation and Performance Evaluation

We implement MobiFish on a Google Nexus 4 smartphone running the Android 4.2 OS. We modify the source code of the Android system, so that it is able to support MobiFish. The MobiFish scheme could be applied to other mobile platforms as well. To evaluate the effectiveness and performance of MobiFish, we conduct experiments for WebFish, AppFish, and AccountFish, respectively.

A. Experiments With WebFish

In the process of evaluating WebFish, we were not able to collect enough phishing web pages specified for mobile platform. Instead, we randomly picked up 100 phishing URLs from PhishTank.com in 2013. Although all the phishing URLs have been blocked by PC browsers like Chrome, they are accessible through mobile browsers (including both Android's built-in browser and Chrome for Android). This highlights the significance of WebFish, which can provide web phishing defense for mobile OSs. In our experiments, WebFish can effectively mark all the phishing URLs as dangerous and can warn the user. Fig. 10(a) and (c) shows that WebFish displays alertness because it is not able to find the SLD inside the mobile phishing login pages of "*Bank of America*" and "*AT&T*."

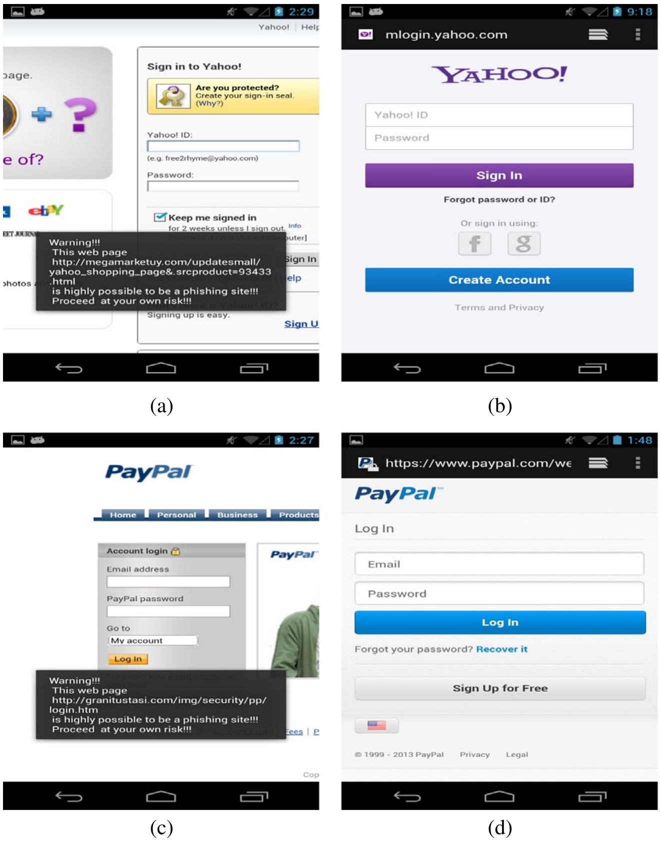
Meanwhile, we have two observations for the conventional (PC) phishing web pages. First, a large number of them are in high similarity to their legitimate counterparts, and the brand names or company logos are close to the input forms. Second, when loading large conventional web pages, mobile browsers often display the area that contains the input form, instead of displaying an overview of the entire web page.


Bank of America
Confirm Your Online Banking Details and Personal Information
Your Online Banking Information
Online ID
ATM or Check Card PIN
Credit/Debit Card
Bank Account
https://secure.bankofameric
Bank of America
Sign In
Enter Online ID:
Save this online ID
Sign In
Privacy & Security
Secure Area
© 2013 Bank of America Corporation. All rights reserved.
Bank of America, N.A. Member FDIC. Equal Housing Lender.
at&t
AT&T Support
To access att.net...
Sign in now!
Email
Password
Forgot Password?
Keep me signed in
https://m.att.com/myatt/un/
myAT&T
Tap into convenience
User ID
Password
Remember Me
Forgot User ID? Forgot Password?
Login
Full Site | Legal | Privacy Policy Updated 6-29-13 | Español
(a)
(b)
(c)
(d)

Warning!!!
This web page
http://christianinsights.com.au/bankofamerica/Boabankgood/Boa/Boa/Boa/index.html
is highly possible to be a phishing site!!!
Proceed at your own risk!!!

Warning!!!
This web page
http://megamarketuy.com/updatebong/Inderxx.htm
is highly possible to be a phishing site!!!
Proceed at your own risk!!!

Fig. 10. Experiments with WebFish on mobile phishing login page. (a) BOA phishing login page. (b) BOA official login page. (c) AT&T phishing login page. (d) AT&T official login page.



Warning!!!
This web page
http://megamarketuy.com/updatesmall/yahoo_shopping_page&.srcproduct=93433.html
is highly possible to be a phishing site!!!
Proceed at your own risk!!!

Warning!!!
This web page
http://granitustasi.com/img/security/pp/login.htm
is highly possible to be a phishing site!!!
Proceed at your own risk!!!

Fig. 11. Experiments with WebFish on conventional phishing login page. (a) Yahoo phishing login page. (b) Yahoo official login page. (c) PayPal phishing login page. (d) PayPal official login page.

Fig. 11(a) and (c) shows the phishing login pages of Yahoo and PayPal. As we can see, the brand name Yahoo and PayPal logo appear more than once in the input form area that is presented to the user. Both of them are reported as phishing sites since WebFish cannot find their SLD in the screenshot.

To evaluate the performance of WebFish on legitimate web pages, we use the URLs of the corresponding official login web pages for comparison. Figs. 10(b), (d), 11(b), and (d) are the official mobile login pages; WebFish successfully verifies the validity of these pages, and no warning is generated. WebFish's ability to verify the legitimate *AT&T* web page shows that the MWL scheme works for company websites with different brand names and SLDs. The 19 corresponding legitimate mobile login web pages can prove WebFish's ability in verifying legitimate web pages. Table I summarizes the testing results of phishing URLs (columns 1–3) and legitimate URLs (columns 4 and 5). The "MWL" behind legitimate SLD name means that the MWL is used to convert the SLD to the brand name.

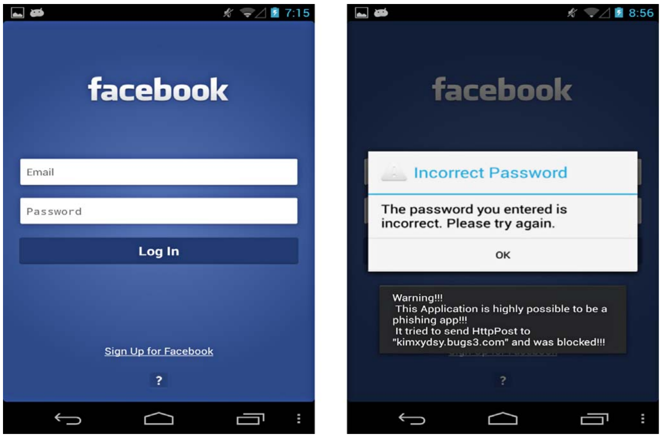
Table I shows that 1) WebFish is able to find key features of phishing web pages for all tested phishing URLs and 2) WebFish achieves a 100% verification rate of legitimate URLs. The results demonstrate the effectiveness of WebFish in detecting mobile phishing sites.

B. Experiments With AppFish

By the time that we conduct experiments with AppFish, there are only a few reported phishing applications, and none of them

TABLE I
SUMMARY OF TESTED URLS

Website	Phishing Samples	Phishing Feature Found	Legitimate Mobile Login SLD	Verification of Legitimate URLs
Amazon	6	100%	amazon	✓
AOL	3	100%	aol	✓
Apple	5	100%	apple	✓
AT&T	2	100%	att (MWL)	✓
Bank of America	10	100%	bankofamerica	✓
Barclays	4	100%	barclays	✓
Chase	5	100%	chase	✓
Citi	4	100%	citibank (MWL)	✓
Ebay	8	100%	ebay	✓
Facebook	5	100%	facebook	✓
Hotmail	2	100%	live (MWL)	✓
HSBC	8	100%	hsbc	✓
Microsoft	1	100%	live (MWL)	✓
NAB	1	100%	nab	✓
NatWest	3	100%	nwolb (MWL)	✓
PayPal	12	100%	paypal	✓
Vodafone	4	100%	vodafone	✓
Wells Fargo	7	100%	wellsfargo	✓
Yahoo	10	100%	yahoo	✓
Total:	100	100%	Tot: 19	100%



Incorrect Password

The password you entered is incorrect. Please try again.

Warning!!!
This Application is highly possible to be a phishing app!!!
It tried to send HttpPost to "kimxydsy.bugs3.com" and was blocked!!!

(a) (b)

Fig. 12. Experiments with AppFish. (a) Fake Facebook app. (b) Warning of phishing app.

is available online. To test the effectiveness of AppFish, we develop two sample phishing applications. Fig. 12(a) shows the login interface of the fake Facebook apps that we developed. Most users are not able to discern its difference from the legitimate Facebook app. Our first phishing application appears as a "repackaged" Facebook app. The second one hijacks the real Facebook app. It can cover the real Facebook login interface in a single window switching slot; hence, the user cannot notice that, in fact, two apps have been launched. When the user clicks the "Log In" button, the fake apps send the credentials to us by HttpGet, HttpPost, socket, SMS, and e-mail, respectively. Meanwhile, a notice window is displayed, informing the user of an incorrect password, and prompts another try. However, when AppFish is running, it is able to block all the transmissions and warn the user about the phishing attempts. Fig. 12(b) (lower part) shows the warning generated by AppFish for the phishing attack.

C. Experiments With AccountFish

The persistent account registry attack is a new class of phishing attacks that we have discovered. As far as we know, there is no such phishing app reported. Hence, we evaluate the effectiveness of AccountFish against the three types of account phishing attacks, using the demo apps that we developed.

Fig. 13(a) presents a type-A account phishing attack, in which a "Greedy Snake" app intends to register a Twitter account. Fig. 13(b) shows a sample type-B attack, i.e., the suspicious "Game Center" app wants to register a Twitter account while hiding itself from the main menu. Both attacks are successfully detected when trying to register the accounts whose labels are inconsistent with their app names. To illustrate the defense to the type-C attack, we develop a "repackaged" Twitter app. It is also detected by AccountFish when trying to send out the credentials [see Fig. 13(c)].

D. Overhead Evaluation

We have validated the efficacy of our proposed three subschemes through the aforementioned experiments. Next, we evaluate the usability (performance) by measuring the execution time of the MobiFish scheme. There are two major techniques used in MobiFish: searching the SLD in the text extracted from the screenshot and blocking the SMS and socket connections for a certain period of time. The SLD searching is based on the OCR technique, which is considered much more time consuming. Hence, we evaluate the delay overhead of OCR-based techniques and other non-OCR techniques separately. The results are presented in Table II.

The OCR-based techniques can be roughly divided into three phases: taking a screenshot, extracting text from the screenshot, and searching if the SLD exists in the text. Since the SLD searching technique is applied in all the three subschemes, our testing samples include 1) the 13 official websites listed in Table I and 13 corresponding phishing web pages and 2) ten popular benign apps that support persistent account and the phishing apps developed by us. The average execution time of the three phases are 0.015, 3.206, and 0.094 s, respectively. As we can see, the OCR extraction phase holds 96.7% of the delay overhead (3.206 out of 3.315 s). Regarding the non-OCR techniques, we tested the SMS and socket connection blocking for a period of time. The app samples used are the same ten popular benign apps and the phishing apps previously mentioned. The delay overhead is very low (0.018 s) because we only need to decide whether the blocking period has expired through a single comparison of two timestamps.

Note that the aforementioned phishing detection techniques are performed in parallel to the normal functions (e.g., web page and app authentication); hence, the user experience will not be influenced. When the checking starts, a toast (a quick message) is displayed to notify the user. Users are suggested to submit the credentials after the checking is done. We believe that, in most cases, the checking can be finished before a mobile user inputs the credentials. Meanwhile, we seek to improve the OCR technique, so that the extraction process can be expedited.

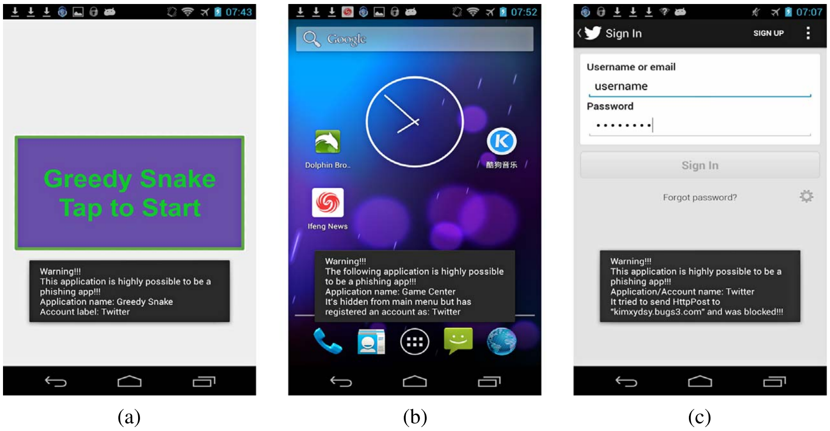
VII. Related Work

A. Conventional Phishing Web Page Detection

Web users have been suffering from phishing attacks since their first appearance in 2003. Researchers have proposed many solutions (such as alert protection and phishing detection) to defend against phishing attacks.

Alert protection is a simple notification when a user is entering sensitive information. Kirda and Kruegel proposed *AntiPhish* [12], which tracks the sensitive information of a user and generates warnings whenever the user attempts to give away this information to a website that is considered untrusted. However, this scheme cannot automatically check and detect phishing attacks. Instead, users have to judge by themselves after being warned.

In addition, many phishing detection tools have been designed for phishing on PC web pages. Based on the methods used, they can be generally categorized into two groups: heuristics schemes and blacklist schemes. Heuristics schemes outperform blacklist schemes since they can deal with new phishing sites without having to wait for an update. Usually, heuristics schemes for phishing detection utilize other



Warning!!!
This application is highly possible to be a phishing app!!!
Application name: Greedy Snake
Account label: Twitter

Warning!!!
The following application is highly possible to be a phishing app!!!
Application name: Game Center
It's hidden from main menu but has registered an account as: Twitter

Warning!!!
This application is highly possible to be a phishing app!!!
Application/Account name: Twitter
It tried to send HttpPost to "kimxydsy.bugs3.com" and was blocked!!!

Fig. 13. Experiments with AccountFish. (a) Type-A attack. (b) Type-B attack. (c) Type-C attack.

TABLE II
AVERAGE EXECUTION TIME OF THE MOBIFISH SCHEME

Techniques	Phases	Execution time (s)	
OCR-based technique	Taking screenshot	0.015	3.315
	OCR extraction	3.206	
	SLD searching	0.094	
Non-OCR technique	Timestamp comparison	0.018	

techniques such as machine learning techniques [13]–[15] and a search engine [15], [16]. CANTINA [16] is a content-based approach to detecting phishing websites, and it adopts term frequency–inverse document frequency information retrieval algorithms. Garera *et al.* [13] proposed a heuristics-based scheme, which identified several generic features of phishing URLs, and used these features in a logistic regression classifier. CANTINA+ [15] is a comprehensive feature-based solution for web page phishing, which combines machine learning and search engine techniques. However, existing heuristics used in phishing detection are all based on features extracted from the HTML source code. As we have shown in Section III, HTML source code should not be trusted since it may not reflect the actual content presented to users.

Based on the assumption that the most spoofing phishing sites are those whose visual appearances look identical or very similar to authentic sites [17], [18], several similarity-based phishing detection approaches are proposed. SpoofGuard [19] uses URLs, images, links, and domain names to check the similarity between a given page and the pages previously stored. Afroz and Greenstadt proposed PhishZoo [20] that uses the profiles of trusted websites' appearances built with fuzzy hashing techniques to detect phishing. PhishZoo makes profiles of sites that consist of fuzzy hashes of several common content elements (e.g., URL, images, most used texts, HTML codes, script files, etc.), which are related to the structure and appearance of the sites. They further enhanced their phishing detection scheme by adding displayed images into profiles and utilizing a scale-invariant feature transform image-matching algorithm [8]. However, similarity-based approaches also depend on HTML source code and cannot detect phishing sites with different appearances.

GoldPhish [9] utilizes the OCR technique for phishing detection in PC browsers. OCR is used to extract text from images found in web pages (e.g., the company logo), and then, it is compared to the top-ranked domains from Google's search service. However, OCR performance on PC is demonstrated to be limited in both speed and accuracy. Our lightweight scheme works with mobile browsers, and it does not depend on external search engines.

B. Mobile Phishing Detection

Mobile phishing attacks are emerging as a significant threat for mobile users. Niu *et al.* [4] discussed the weakness of mobile browsers caused by the hardware limitation of mobile devices. Felt and Wagner [6] examined the mobile phishing threats by detailing several phishing attack models during control transfers. Both works give some suggestions on phishing mitigation. Niu *et al.* [4] advised redesigning the browsers to make the origin and authenticity of the site more apparent to users. However, it is very difficult to add more features to the mobile UI due to the limited screen size, and even if they are added, some web users will still ignore the identifier [21]. Felt and Wagner [6] proposed to add an always-present identity bar that displays the name of the current foreground application or the domain name of the current web page. Bianchi *et al.* [22] implemented an identity indicator for apps in the system navigation bar, in which extended-validation HTTPS infrastructure is used to validate the app developers. Marforio *et al.* [23] applied personalized security indicators (an image chosen by the user that is displayed in the login UI) to mobile apps. However, all these indicator-based approaches require the user to make the final decision.

Another group of phishing defense techniques employs a unified and trusted login UI for apps. ScreenPass [24] provides a trusted software keyboard, which allows users to specify their password domains (i.e., to tag passwords) together with the credentials. The OCR is used to ensure that passwords are entered only through the trusted software keyboard. This approach needs the user to switch to the secure keyboard when entering the password, tag the password, and make the final decision, which may greatly degrade the user experience. VeriUI [25] utilizes an attested login, which augments user credentials with a certificate about the software and hardware that handled the credentials. However, this work requires not only the user effort but modifications to the client apps as well.

Moreover, a proxy service is designed in [4], which performs antiphishing filtering against the URLs, page content, or user context. However, it has to be downloaded and configured manually in the browser. Users also need to be able to authenticate the identity of the proxy (attackers can also set up fake proxies). Hou and Yang [26] developed a defense scheme, which loads a hook into the iOS so that the system interrupts the user, when sensitive information is being entered into applications not in the whitelist, and prompts the user to decide whether to continue or not. However, this idea is quite similar to *AntiPhish* [12], which only gives a warning of credential rendering, instead of phishing vulnerability. Cooley *et al.* proposed Trusted Activity Chains [27] to protect activities from spoofing preventions. However, it is the developer's responsibility to annotate the chain of activities that should not be interrupted. This means that existing apps are not protected, and the developers may not assume the extra burden of annotation. Our previous work [28] proposed the WebFish and AppFish schemes. In this paper, we present the new persistent account phishing attacks, which has been neglected by existing works. We resolve this vulnerability with the AccountFish scheme.

Our work differs from previous works in three ways. 1) MobiFish is a completely automated defense scheme, i.e., users do not need to make the final decision. Although it is users who finally remove the phishing app, the user effort is trivial. They do not need to explicitly make the decision at all, since the only explanation for the login failure (with correct credentials) is a phishing attack. 2) No change is required to the browser/app/website's UIs, i.e., MobiFish is compatible with all existing websites and apps (no developer effort is needed). 3) The phishing attacks targeted at the persistent account are discovered and handled by the AccountFish scheme.

In addition, mobile phishing attacks could also be in the form of e-mails or SMS. Phishing e-mails usually request users to click a link to a fake website, where the user is prompted to enter login credentials [29], [30]. The SMS phishing attacks (SMiShing) [31], [32] usually trick users into visiting a fraudulent website or calling a phishing number, where the victims are enticed into providing credentials. The fraudulent websites could be defended by WebFish. However, the detection of the phishing voice calls is beyond the scope of this paper. Most voice phishing (Vishing) uses the voice over internet protocol (VoIP) technique, in which the phone number is dynamically generated; we left this part for future work.

VIII. Conclusion

In this paper, we have studied the important issue of mobile phishing detection. We proposed MobiFish, which is a novel automated phishing defense scheme for mobile platforms. We identified the weaknesses of the heuristics-based antiphishing schemes that highly rely on the HTML source code of web pages. MobiFish resolves this issue by using OCR, which can accurately extract text from the screenshot of the login interface so that the claimed identity can be verified. Compared to existing OCR-based antiphishing schemes (designed for PC only), MobiFish is lightweight because it works without using external search engines or machine learning techniques. In addition, MobiFish can also detect the app phishing attacks and account phishing attacks. We implemented MobiFish on a Google Nexus 4 smartphone running the Android 4.2 OS. Our evaluation demonstrated that MobiFish can effectively detect and defend against mobile phishing attacks.

References

[1] "Phishing activity trends report," Anti-Phishing Working Group, Cambridge, MA, USA, 2006.

[2] L. F. Cranor, S. Egelman, J. I. Hong, and Y. Zhang, "Phinding phish: Evaluating anti-phishing tools," in *Proc. 14th Annu. NDSS*, Feb. 2007, pp. 1–16.

[3] "Mobile phishing: A problem on the horizon," Trend Micro, Tokyo, Japan, 2012.

[4] Y. Niu, F. Hsu, and H. Chen, "iPhish: Phishing vulnerabilities on consumer electronics," in *Proc. 1st Conf. Usability, Psychol., Security*, 2008, pp. 10:1–10:8.

[5] C. Karlof, J. D. Tygar, and D. Wagner, "Conditioned-safe ceremonies and a user study of an application to web authentication," in *Proc. 5th SOUPS*, 2009, p. 38:1.

[6] A. P. Felt and D. Wagner, "Phishing on mobile devices," in *Proc. W2SP*, 2011, pp. 1–10.

[7] A. Bergholz *et al.*, "New filtering approaches for phishing e-mail," *J. Comput. Security*, vol. 18, no. 1, pp. 7–35, Jan. 2010.

[8] S. Afroz and R. Greenstadt, "PhishZoo: Detecting phishing websites by looking at them," in *Proc. 5th IEEE ICSC*, 2011, pp. 368–375.

[9] M. Dunlop, S. Groat, and D. Shelly, "GoldPhish: Using images for content-based phishing analysis," in *Proc. 5th ICIMP*, 2010, pp. 123–128.

[10] Tesseract OCR. [Online]. Available: http://code.google.com/p/tesseract-ocr/

[11] "How Android Users Interact With Their Phones," Yahoo Aviate, 2014. [Online]. Available: http://yahooaviate.tumblr.com/image/95795838933

[12] E. Kirda and C. Kruegel, "Protecting users against phishing attacks with AntiPhish," in *Proc. 29th Annu. Int. COMPSAC*, 2005, pp. 517–524.

[13] S. Garera, N. Provos, M. Chew, and A. D. Rubin, "A framework for detection and measurement of phishing attacks," in *Proc. ACM WORM*, 2007, pp. 1–8.

[14] Y. Pan and X. Ding, "Anomaly based web phishing page detection," in *Proc. 22nd ACSAC*, 2006, pp. 381–392.

[15] G. Xiang, J. Hong, C. P. Rose, and L. Cranor, "Cantina+: A feature-rich machine learning framework for detecting phishing web sites," *ACM Trans. Inf. Syst. Security*, vol. 14, no. 2, pp. 21:1–21:28, Sep. 2011.

[16] Y. Zhang, J. I. Hong, and L. F. Cranor, "CANTINA: A content-based approach to detecting phishing web sites," in *Proc. 16th Int. Conf. WWW*, 2007, pp. 639–648.

[17] J. S. Downs, M. B. Holbrook, and L. F. Cranor, "Decision strategies and susceptibility to phishing," in *Proc. 2nd SOUPS*, 2006, pp. 79–90.

[18] M. Jakobsson, A. Tsow, A. Shah, E. Blevis, and Y.-K. Lim, "What instills trust? A qualitative study of phishing," in *Proc. 11th Int. Conf. Financial Cryptography 1st Int. Conf. Usable Security*, 2007, pp. 356–361.

[19] N. Chou, R. Ledesma, Y. Teraguchi, and J. C. Mitchell, "Client-side defense against web-based identity theft," in *Proc. 11th Annu. NDSS*, Feb. 2004.

[20] S. Afroz and R. Greenstadt, "PhishZoo: An Automated Web Phishing Detection Approach Based on Profiling and Fuzzy Matching," Drexel Univ., Philadelphia, PA, USA, Tech. Rep. DU–CS–09–03, 03 2009.

[21] R. Dhamija, J. D. Tygar, and M. Hearst, "Why phishing works," in *Proc. SIGCHI CHI*, Apr. 2006, pp. 581–590.

[22] A. Bianchi *et al.*, "What the app is that? Deception and countermeasures in the Android user interface," in *Proc. IEEE Symp. SP*, May 2015, pp. 931–948.

[23] C. Marforio, R. J. Masti, C. Soriente, K. Kostiainen, and S. Capkun, "Personalized security indicators to detect application phishing attacks in mobile platforms," *CoRR abs/1502.06824*, 2015.

[24] D. Liu, E. Cuervo, V. Pistol, R. Scudellari, and L. P. Cox, "ScreenPass: Secure password entry on touchscreen devices," in *Proc. 11th Annu. Int. Conf. MobiSys*, 2013, pp. 291–304.

[25] D. Liu and L. P. Cox, "VeriUI: Attested login for mobile devices," in *Proc. 15th Workshop Mobile Comput. Syst. Appl. HotMobile*, 2014, pp. 1–6.

[26] J. Hou and Q. Yang, "Defense against mobile phishing attack," *Comput. Security Course Project*, 2012. [Online]. Available: http://www-personal.umich.edu/~yangqi/pivot/mobile_phishing_defense.pdf

[27] B. Cooley, H. Wang, and A. Stavrou, "Activity spoofing and its defense in Android smartphones," *Appl. Cryptography Netw. Security*, vol. 8479, pp. 494–512, 2014.

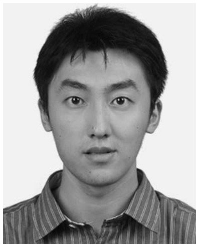
[28] L. Wu, X. Du, and J. Wu, "MobiFish: A lightweight anti-phishing scheme for mobile phones," in *Proc. 23rd ICCCN*, Aug. 2014, pp. 1–8.

[29] A. Almomani, B. Gupta, S. Atawneh, A. Meulenberg, and E. Almomani, "A survey of phishing e-mail filtering techniques," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 4, pp. 2070–2090, 4th Quart., 2013.

[30] J. Wang, T. Herath, R. Chen, A. Vishwanath, and H. Rao, "Phishing susceptibility: An investigation into the processing of a targeted spear phishing email," *IEEE Trans. Prof. Commun.*, vol. 55, no. 4, pp. 345–362, Dec. 2012.

[31] A. Eshmawi and S. Nair, "Smartphone applications security: Survey of new vectors and solutions," in *Proc. AICCSA*, May 2013, pp. 1–4.

[32] A. Kang, J. Dong Lee, W. Kang, L. Barolli, and J. Park, "Security considerations for smart phone smishing attacks," *Adv. Comput. Sci. Appl.*, vol. 279, pp. 467–473, 2014.



Longfei Wu received the B.E. degree in communication engineering from the Beijing University of Posts and Telecommunications, Beijing, China, in 2012. He is currently working toward the Ph.D. degree with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA, USA, under the supervision of Dr. X. Du.

His research interests include smartphone security and wireless networks.

Xiaojiang Du (SM'07) received the B.S. and M.S. degrees from Tsinghua University, Beijing, China, in 1996 and 1998, respectively, and the M.S. and Ph.D. degrees from the University of Maryland, College Park, MD, USA, in 2002 and 2003, respectively, all in electrical engineering.

Between August 2004 and July 2009, he was an Assistant Professor with the Department of Computer Science, North Dakota State University, Fargo, ND, USA. He is currently an Associate Professor with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA, USA. His research interests include security, cloud computing, wireless networks, computer networks, and systems. He has published over 150 journal and conference papers in these areas.

Dr. Du is a Life Member of ACM. He has received more than $3M in research grants from the U.S. National Science Foundation (NSF), the Army Research Office, the Air Force Research Laboratory, NASA, the Commonwealth of Pennsylvania, and Amazon. He received the Excellence in Research Award from North Dakota State University in May 2009. He serves on the editorial boards of four international journals. He will serve as the Lead Chair of the 2015 Communication and Information Security Symposium of the IEEE International Conference on Communications (ICC) and as a Cochair of the 2015 Mobile and Wireless Networks Track of the IEEE Wireless Communications and Networking Conference (WCNC). He was the Chair of the Computer and Network Security Symposium of the IEEE/ACM International Wireless Communication and Mobile Computing Conference during 2006–2010. He is (was) a Technical Program Committee Member of several premier ACM/IEEE conferences, such as the Conference on Computer Communications (INFOCOM, 2007–2015), the Symposium on Integrated Management (IM), the Network Operations and Management Symposium (NOMS), the International Conference on Communications (ICC), the Global Communications Conference (GLOBECOM), the Wireless Communications and Networking Conference (WCNC), the Broadband Wireless Networking Symposium (BroadNet), and the International Performance Computing and Communications Conference (IPCCC).

Jie Wu (F'09) received the B.S. degree in computer engineering and the M.S. degree in computer science from Shanghai University of Science and Technology, Shanghai, China, in 1982 and 1985, respectively. He received the Ph.D. degree in computer engineering from Florida Atlantic University, Boca Raton, FL, USA, in 1989.

He is the Chair and a Laura H. Carnell Professor of the Department of Computer and Information Sciences, Temple University, Philadelphia, PA, USA. He is also an Intellectual Ventures Endowed Visiting Chair Professor with the National Laboratory for Information Science and Technology, Tsinghua University, Beijing, China. Prior to joining Temple University, he was a Program Director with the National Science Foundation and a Distinguished Professor with Florida Atlantic University, Boca Raton, FL, USA. He regularly publishes in scholarly journals, conference proceedings, and books. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications.

Dr. Wu is a China Computer Federation (CCF) Distinguished Speaker. He was an IEEE Computer Society Distinguished Visitor, an ACM Distinguished Speaker, and the Chair for the IEEE Technical Committee on Distributed Processing. He received the 2011 CCF Overseas Outstanding Achievement Award. He serves on several editorial boards, including the IEEE TRANSACTIONS ON SERVICE COMPUTING and the *Journal of Parallel and Distributed Computing*. He has served as the General Cochair/Chair for the 2006 IEEE International Conference on Mobile Ad Hoc and Sensor Systems (MASS), the 2008 IEEE International Parallel and Distributed Processing Symposium (IPDPS), the 2013 IEEE International Conference on Distributed Computing Systems (ICDCS), and the 2014 ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc), as well as Program Cochair for the 2011 IEEE Conference on Computer Communications (INFOCOM) and the 2013 CCF China National Computer Congress (CNCC). He currently serves as the General Chair for ACM MobiHoc 2014.