Abusing Notification Services on Smartphones for Phishing and Spamming

Zhi Xu
Department of Computer Science & Engineering
Pennsylvania State University
zux103@cse.psu.edu

Sencun Zhu
Department of Computer Science & Engineering
College of Information Sciences & Technology
Pennsylvania State University
szhu@cse.psu.edu

ABSTRACT

Notification service is a popular functionality provided by almost all modern smartphone platforms. To facilitate customization for developers, many smartphone platforms support highly customizable notifications, which allow the third party applications to specify the trigger events, the notification views to be displayed, and the allowed user operations on the notification views.

In this paper, we show that notification customization may allow an installed trojan application to launch phishing attacks or anonymously post spam notifications. Through our studies on four major smartphone platforms, we show that both Android and BlackBerry OS are vulnerable under the phishing and spam notification attacks. iOS and Windows Phone allow little notification customization, thus launching the phishing and spam attacks will expose the identity of the trojan application. Attack demonstrations on all platforms are presented.

To prevent the phishing and spam notification attacks while still allowing notification customization, we propose a *Semi-OS-Controlled* notification view design principle and a *Notification Logging* service. Moreover, to protect applications from fraudulent views, we propose a view authentication framework, named *SecureView*, which enables the third party applications to add the authentication image and text to their sensitive views (e.g. the account login view). The implementation and demonstrations of proposed defense approaches on Android are also presented in the paper.

1. INTRODUCTION

Phishing and spam attacks are two of the most successful and profitable attacks [1, 2]. Both attacks have achieved great success through traditional medias, such as email, web, and instant messaging. With the fast growth of smartphone markets, we have seen more and more attempts to launch phishing and spam attacks on popular smartphone platforms [3, 4, 5]. According to the statistical results in *Trusteer report* [6], smartphone users are more vulnerable to phishing and spam attacks due to the constrained UI on smartphones. However, most existing phishing and spam attacks on smartphones still rely on traditional media, such as SMS [7], mobile email [6] and mobile web [4]. Studies are needed regarding launching phishing and spam attacks with new features provided by modern smartphone platforms.

In this paper, we examine the notification service of four most popular smartphone platforms [8], and investigate the feasibility for an installed trojan application to distribute phishing notifications or anonymous spam notification on these platforms. The selected smartphone platforms include Android (version 2.3 and 4.0), BlackBerry OS (version 7), iOS (version 5), and Windows Phone (version 7.1). According to comScore's latest report [8], the selected four smartphone platforms cover 98% share of the U.S. smartphone market. Therefore, our results should have immediate impacts on the current smartphone market.

Table 1: A summary of experimental results(✓means feasible and ✗means infeasible)

Platforms	Send Anonymous Notifications	Navigate To Fraudulent View
Android 2.3	✓	✓
Android 4.0	✓	✓
BlackBerry OS 7	✓	✓
iOS 5	✗	✗
Jailbroken iOS 5	✓	✓
Windows Phone 7.1	✗	✓

Notification service is a system service provided by the smartphone platform to both system and third party applications. It allows a notification sender application to notify the smartphone user with a notification view when a trigger event happens. Common notifications include system update notifications, battery shortage notifications, and calender reminder notifications.

Despite the differences in the design and implementation of different smartphone platforms, a notification can be described by three elements: the *Trigger Event* that fires the notification; the *Notification View* that is displayed on the screen; and the *Allowed User Operations* that defines the invoked actions when a user operates on the displayed notification view.

To facilitate the application development, many smartphone platforms allow the third party applications to send customized notifications. For example, Android and BlackBerry allow third party applications to submit a customized view object in the notification view. iOS and Windows Phone, being less customizable, only allow submitting text content. On all these four smartphone platforms, a sender application can trigger notifications when it is not running in the foreground.

Notification customization is widely endorsed by the developer communities. However, an installed third party application may abuse the notification customization to send anonymous notifications or fraudulent notifications. The anonymous notifications can be used to distribute unsolicited advertisements, and the fraudulent notifications can be used to launch phishing attacks against other installed applications. For example, a trojan application may generate a fraudulent notification that mimics the Facebook notification and leads the user to a fraudulent login view.

Our studies on Android and BlackBerry OS show that it is feasible for a trojan application to launch both phishing and spam attacks using notification services while hiding among other applications. Through attack demonstrations, we show that a common third party application can be easily modified to become a trojan application capable of launching the proposed phishing and spam notification attacks. According to the market share report [8], the combined market share of Android and BlackBerry OS is about 63.4% in US. On iOS and Windows Phone, due to their limited cus-

tomizations, launching the phishing and spam attacks will expose the identity of a trojan application. A summary of our experimental results is shown in Table 1. Further, approaches for stealthy content distribution are discussed to help the proposed trojan applications bypass the application review process of the application store.

To prevent the phishing and spam attacks while still allowing notification customization, we propose three defense approaches.

First of all, for smartphone platforms, we propose a new *Semi-OS-Controlled* design principle for the notification view. A Semi-OS-Controlled notification view contains a customizable subview controlled by the sender application, and a wrapper frame that is under control of the smartphone platform. To prevent the phishing and spam attacks, the wrapper frame contains the authentication information of the sender application.

Secondly, we propose a notification logging service that records posted notifications with sender authentication information, so that an user can review any suspicious notifications that were posted.

Thirdly, we propose a view authentication framework, named *SecureView*, which enables a third party application to prevent phishing notification attacks by adding an authentication image and text to its sensitive views (e.g. the account login view).

The contributions of paper are summarized as follows:

- To the best of our knowledge, we are the first to discuss the feasibility of launching phishing attacks using the notification service on smartphones.
- We present detailed designs, implementations, and demos on how to launch phishing and spam attacks on four major smartphone platforms.
- We present three feasible defense approaches that can be easily applied on existing smartphone platforms, and present the implementations of the proposed approaches on Android.

We hope our proactive security study will draw attention of smartphone users as well as platform vendors.

2. SMARTPHONE NOTIFICATION SERVICE

2.1 Overview

A *notification* is a message dialog that is shown on the surface of a smartphone screen. To send a notification, the sender application first prepares a notification object and registers a trigger event for it. When the trigger event occurs, this notification will be fired.

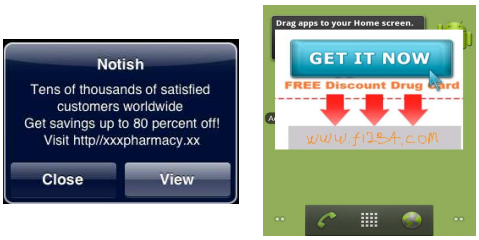
As the notification service is a common service with almost no abuse reported yet, accessing notification service requires no security permissions on most smartphone platforms, such as Android, BlackBerry OS, and Windows Phone.[1]

Despite the differences in design and implementation, notification services on existing smartphone platforms can be categorized into three types (shown in Figure 1 and Table 2):

- *Pop-up Notification* notifies the user by popping up a notification dialog on screen, e.g., the Toast Notification on Android and Dialog Notification on BlackBerry OS.
- *Status Bar Notification* adds a notification view as a status bar at the top of screen, e.g., the Message List on BlackBerry OS 7, the Status Bar Notification on Android, and the Notification Center on iOS 5.
- *Icon Notification* shows the notification by making changes on the icon of a sender application in the main menu, e.g., the Badge notification on iOS 5 and the Icon Update function on BlackBerry OS 7.

In this paper, we focus on launching phishing and spam attacks with pop-up notification and status bar notification. The icon notification is not applicable because it will expose the identity of a sender application.

[1]iOS requires the third party applications to explicitly request permissions from the user for push notification service after installations



(1) Notification Alert on iOS 4 (OS-Controlled)

(2) Toast on Android (App-Controlled)

Figure 2: A comparison of OS-Controlled and App-Controlled notification views

2.2 Notification Customization

For notification customization, a sender application specifies a notification from three aspects: (1) the trigger event that will fire this notification, (2) the content to display in this notification, and (3) the allowed user operations on this notification.

Trigger Event: The trigger event is an event that fires the notification. It could be a system event (e.g. scheduled alarms), or an application event generated by an application service running in the background. The flexibility on notification triggering allows a trojan application to fire anonymous notifications when it is not running in the foreground. For example, on Android, a sender application can fire notifications whenever the screen is turned on with the *Intent.ACTION_SCREEN_ON* intent broadcasted by OS.

Notification View: The *Notification View* represents the displayed view of a notification. It carries the content that the sender application wants to show on the screen, such as a text message, an image, or even a complex subview. By the control of displayed contents, the notification views in existing notification services can be categorized into two groups, *OS-Controlled* and *App-Controlled*.

In *OS-Controlled* notification views, the layout of a notification view is fixed and the sender application can only pass a text message to display. For example, in the Notification Alert on iOS 5 (shown in Figure 2(1)), the only thing that the sender application can control is the text message shown in the notification view.

In *App-Controlled* notification views, a sender application is allowed to manipulate the whole notification view, including the layout of view. Typical App-Controlled notification views include the Toast and Status Bar notification on Android, and the Dialog on BlackBerry OS 7. In Figure 2(2), we present a toast notification on Android with a single image in the notification view. Compared to the Figure 2(1), this customized toast notification hides the identity of sender application (i.e., "Notish") and displays an unsolicited advertisement by customizing the notification view.

Our proposed attacks rely on the App-Controlled notification views to send fraudulent phishing notifications and anonymous spam notifications without exposing the identity of a trojan application.

Allowed User Operations: The allowed user operations include operations which the user can perform on the displayed notification, as well as the invoked actions of these operations. Due to the limited UI of smartphones, the allowed user operations are usually restricted. One simple example of user operation would be: *dismissing the notification when the notification view is tapped*. More advanced actions include launching local applications (e.g., by firing an Intent on Android), opening the web browser with a specified URL address (e.g., by calling the web browser API on BlackBerry OS 7), and reopening the sender application (e.g., when a user clicks the View button on the notification alert on iOS 5).

On Android and BlackBerry OS, a sender application can specify all the allowed user operations of a notification. To the opposite, on iOS, no customization is allowed on the allowed user operations. On Windows Phone, the sender application can only specify the view that will be navigated to.

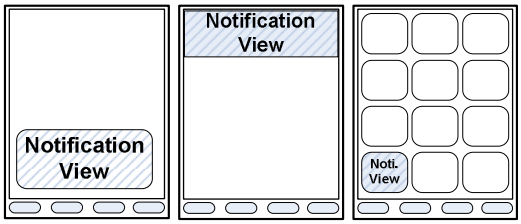


Figure 1: Three types of notification services

Platforms	Pop-up Notification	Status Bar Notification	Icon Notification
Android 2.3	Toast	Status Bar	✗
Android 4.0	Toast	Status Bar	✗
BlackBerry OS 7	Dialog	Message List	Icon
iOS 5	Alert	Notification Center	Badge
Windows Phone 7.1	ShellToast	✗	Tile

Table 2: Notification services on smartphone platforms

The control of allowed user operations is a prerequisite for the success of proposed phishing and spam notification attacks. In spam notification attacks, a trojan application specifies the allowed user operations to cover up its identity from being exposed to the user. In phishing notification attacks, besides hiding its identity, a trojan application navigates to the fraudulent views (e.g. a fake login view) when the user taps the notification view.

3. PROPOSED PHISHING AND SPAM NOTIFICATION ATTACKS

In this section, we study the feasibility of sending fraudulent phishing notifications and anonymous spam notifications using an installed trojan application on a smartphone. Fraudulent notifications mimic the genuine notifications of target applications. Similar to adware, spam notifications contain unsolicited advertisements, such as commercial texts and images.

3.1 Overview of Proposed Attacks

To facilitate our discussion, we name our proposed trojan application *Notish* and its application icon is totally different from the Facebook application icon.

3.1.1 Phishing Notification Attack

In the phishing notification attack, the attack goal of *Notish* is to steal the user's credentials for other installed applications, for example, the user's account id and corresponding password of an installed mobile banking application. In our demonstrations, we take the *Facebook* application as our target application, and show how to trick the user to enter her user id and password on a view controlled by *Notish*.

Briefly, *Notish* first sends a fraudulent notification mimicking the notifications generated by the Facebook application. When the user taps on the fraudulent notification, she will be led to a fraudulent login view controlled by *Notish*. The fraudulent login view is a part of *Notish*, but it looks exactly the same as the login view of the Facebook application. The user id and the password entered on the fraudulent login view will be received by *Notish*. When the user clicks the login button, he will then be transferred to either the genuine Facebook application (e.g. on Android), or the genuine Facebook website (e.g. on BlackBerry). Meanwhile, the fraudulent login view, together with *Notish*, will terminate immediately after the transfer in order to hide *Notish*.

3.1.2 Spam Notification Attack

In the spam notification attack, the attack goal of *Notish* is to send spam notifications while hiding its identity from the user. Briefly, *Notish* first sends a spam notification with unsolicited advertisements. When the user taps on the spam notification, this spam notification will dismiss and the user will be led to a spam website in the mobile browser.

However, the spam notification attacks are also more perceptible in that the user will notice the posted unsolicited notifications. If the user can identify the sender application of spam notification, he may uninstall the trojan application and report the abuse to the app store. Although the trojan vendor may introduce new trojan applications after the exposure, the ephemeral trojan application will make the spam attack expensive. A detailed study of detecting notification attacks will be presented in Section 5.

3.2 Attack Flows

We elaborate the attack flow in 3 phases: $plot \rightarrow play \rightarrow cleanup$.

3.2.1 Plot

The plot phase happens when the trojan application is active, running either in the foreground or in background. In this phase, *Notish* specifies the details of the notification to be posted.

For trigger events, *Notish* may set the trigger event of notification at a time point when *Notish* is inactive or running in the background. For example, on Android, *Notish* may use the broadcasted Intent to trigger the notification whenever the screen is turned on; on BlackBerry OS 7, *Notish* can set a scheduled alarm to fire the notification at any time point suitable for *Notish* to hide itself.

Phishing Attacks: To craft the fraudulent notification view, *Notish* will select the target application from installed applications on the current smartphone. To get the list of installed applications, it may use the $PackageManager$ API on Android and $CodeModuleManager$ API on BlackBerry OS. According to the selected target application, *Notish* prepares the fraudulent notification view and the fraudulent login view. For the allowed user operations on the fraudulent notification view and login view, *Notish* has to mimic those operations provided by the target applications. Meanwhile, *Notish* has to make sure that the user will not be able to tell the identity of *Notish* based on the fraudulent notification and fraudulent login view.

Spam Attacks: The unsolicited advertisements are predefined. Receiving spam contents from remote parties enables *Notish* to update the spam content as well as bypassing the application review process in the app store. The detailed benefits and design will be presented in Section 5. Also, *Notish* has to carefully specify all allowed user operations on this spam notification. For example, a conservative option would be dismissing the notification immediately no matter what operation the user will perform. A better option would be leading the user to a dedicated spam website after the user operates on the notification.

3.2.2 Play

When the specified trigger event occurs, the fraudulent phishing or spam notification will be displayed on the screen of smartphone.

Phishing Attacks: After tapping on the notification, the fraudulent login view will be displayed asking the user to enter his user id and password to login. According to [9], asking the user to enter user id and password is common for mobile banking and online social applications. Users will not be surprised about being asked to enter such information.

Notish may also navigate the user to a fraudulent login website in the browser instead of a fraudulent login view. This trick is popular in email phishing attacks and discussed in [4, 5]. However, fraudulent login view is more favorable when the target is an installed application. Fraudulent login website may cause suspicion.

Spam Attacks: After tapping on the spam notification, the notifi-

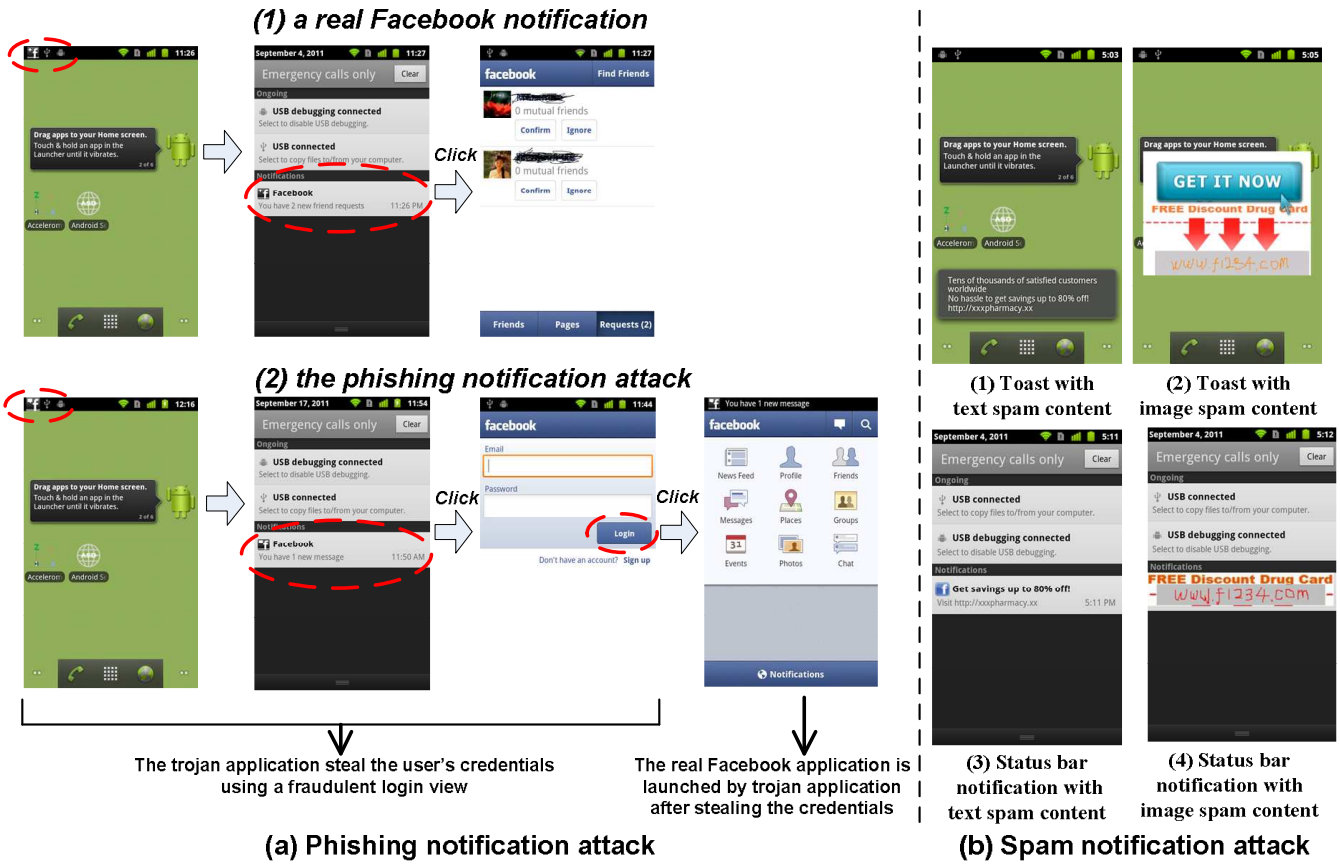


Figure 3: Phishing and spam notification attack scenarios on Android 2.3

cation view will dismiss and the user might be led to a dedicated spam website in the mobile browser.

3.2.3 *Cleanup*

The major task in the cleanup phase is to erase traces of the phishing and spam notification attack so that the suspecting user will not be able to pinpoint *Notish*. In the cleanup phase, actions specified in the plot phase will be taken when user operations are received. For example, the displayed notification will be dismissed immediately; if the notification is fired by a background service of the trojan application, the service will be ended as well.

Phishing Attacks: Note that, when the fraudulent login view is displayed, *Notish* will take the risk of being active and running in the foreground. Thus, cleanup works are needed for both the notification view and the fraudulent login view. A detailed analysis of attack detection will be presented in Section 5.

Spam Attacks: Further, if *Notish* receives the spam contents from its application vendor, it will delete the received spam content as well in the clearnup phase.

3.3 Proposed Attacks on Android

Android is the dominant smartphone platform in the U.S. smartphone market with about 43.7% market share in 2011 Q2, according to [8]. In our study, we consider two releases of Android: Android 2.3 Gingerbread, the current major version, and Android 4.0 Ice Cream Sandwich, the latest version released in October 19, 2011. Both versions have a Toast notification, which is a type of Pop-up notification, and a Status Bar notification.

The proposed phishing and spam notification attacks are successful on both versions of Android. As there is little change on the notification service from Android 2.3 to Android 4.0, in this section we use Android 2.3 for demonstration. Due to the space limit, the demonstration on Android 4.0 is presented in the Appendix A.

3.3.1 *Attack with Toast*

Android allows sending a toast notification with a default layout provided by Android, or with a customized layout provided by the sender application. The default layout only accepts text content and shows no authentication information. The customized layout is under fully control of sender application. As shown in Figure 3(b), the trojan application can successfully send toast without exposing its identity within the notification view.

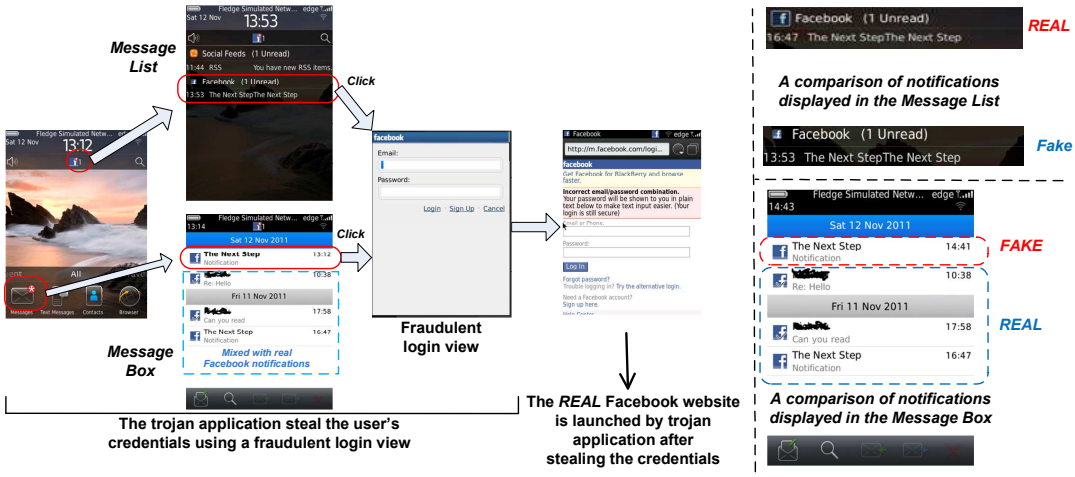
Moreover, the toast notification does not accept user operations, and will be dismissed automatically after display. Thus, the annoyed user cannot identify the sender application based on the displayed toast notification.

3.3.2 *Attack with Status Bar Notifications*

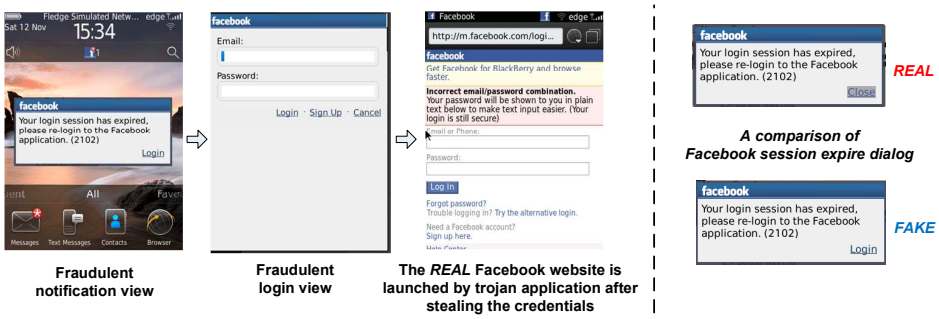
On Android, the status bar notification service provides two types of notification views. One type of notification view is generated with the default layout. The other type of view is assigned directly by the sender application. With the default layout, the sender application is allowed to manipulate the application icon as well as the text content. As shown in Figure 3(b), *Notish* replaces its application icon with a Facebook icon in order to trick the user. With the view assigned by the sender application, the sender application can manipulate the notification view at will. In Figure 3(b), *Notish* displays a single image in the notification view.

For status bar notification, a user is allowed to perform only two types of operations, one is "Clicking the notification", and the other is clicking the "Clear all notifications" button on the screen, which will trigger Intent *contentIntent* and *deleteIntent*, respectively. The trojan application can specify the corresponding actions for these two Intents in the plot phase when creating the spam notification. For example, in our demo of Figure 3(b), when the user clicks on the notification, the spam notification will dismiss, and the default web browser will be launched with the URL of a spam website.

As the notification view and all allowed user operations are under



(a) Phishing notification attacks with Message List



(b) Phishing notification attacks with Dialog

Figure 4: Phishing notification attack scenarios on BlackBerry OS 7

control of the trojan application, a user cannot identify the sender application from the displayed spam notification.

3.3.3 *Attack Demonstration*

Phishing Attacks: In Figure 3, we present the attack flow specifically on Android 2.3, and compare it with the work flow of the genuine Facebook application. The proposed attack uses status bar notification because it is also used by the genuine Facebook application on Android. As the Toast accepts no user operations, it cannot be used for phishing attacks.

As shown in the Figure 3, by customizing the notification view, the trojan application can send a fraudulent notification view that is exactly the same as the real notification from the Facebook application. When the user taps on the fraudulent notification view, the fraudulent login view will be brought to foreground.

The trojan application can launch the real Facebook application after receiving the credentials. Many mobile applications keep the user's session for a long period of time [9]. For example, as observed in [10], the Facebook app on Android will keep the session forever unless the user manually logout the app. If the session is not expired, when the trojan application launches the genuine Facebook application, the genuine Facebook application may navigate to the main menu directly (as shown in Figure 3). This will help further reduce the suspicion of phishing attacks. If the current session has expired, a new login view identical to the fraudulent login view will be shown by the genuine Facebook application. Seeing the new login view might cause little cause suspicion. However, as buttons on the touchscreen are close to each other, it is common for the user to enter wrong passwords by mistake and then be directed to the login view again.

To sum up, three advanced characteristics of the view navigation mechanism on Android are the keys to the success of our phishing notification attack. Firstly, status bar notification view is customizable. Secondly, Android allows a user to be navigated to a specified view belonging to the sender application, such as the fraudulent login view. Thirdly, Android allows an installed application to launch another installed application.

Spam Attacks: In Figure 3(b), we show the demonstrations of spam notification attacks launched by a trojan application. In all demonstrations, we trigger the notification 5 seconds after the screen is turned on using a background service.

3.4 Proposed Attacks on BlackBerry OS

BlackBerry OS is a popular smartphone platform that is famous for its customizable notification service. The BlackBerry OS 7 provides a *Dialog* service for pop-up notification, a *Message List* service for status bar notification, and an *Icon Update* service for icon notification. Here we show that it is feasible to launch phishing notification attacks with either the Dialog or the Message List.

3.4.1 *Attack with Dialog*

Dialog is a system service that pops up a predefined dialog on the screen and waits for user input. To trigger a notification, system events can be applied. For example, in our demo attack, *Notish* keeps a background service which auto-runs on startup. The background service uses a *BackgroundClockListener* to generate the trigger event at a time point specified by *Notish*.

The notification view of Dialog is App-Controlled, which means *Notish* fully controls the contents of notification views. As shown in Figure 5, BlackBerry OS 7 allows filling the Dialog notification view with not only text content by its default layout, but also customized views (e.g., the fraudulent Facebook notification).

Dialog provides richer notification views that allow buttons to be contained within the notification view. However, all user operations on the notification view of Dialog are under the control of a trojan application. Take the "OK" button in Figure 5 as an example, *Notish* can associate it with a button handling function. When the user presses the "OK" button, the Dialog will be closed and the default web browser will be launched with the URL of a spam website. Furthermore, BlackBerry allows specifying responses even from the hard keyboard. In case the user presses any button in the hard keyboard, the Dialog will be closed and the spam website will be brought up in the browser.

3.4.2 Attack with Message List

Message List is a type of status bar notification service that only supports text based notifications. One unique feature of Message List is that, when a new notification is put into the status bar, a copy of notification messages may be saved within the *Message Box*. The same trigger events for Dialog notifications can also be applied to Message List notifications.

The notification views in both the Message List and the Message Box are OS-Controlled with a fixed layout. However, the sender application is allowed to specify almost all the contents within the default layout. As shown in Figure 5, *Notish* can specify the image of indicator in Figure 5 (1), the subject and text content in Figure 5 (2.1), and the icon and text contents in Figure 5 (2.2). The only element that *Notish* cannot manipulate directly is the application icon shown in Figure 5 (2.1). To cover up the real application icon, the *updateIcon* function can be invoked by *Notish* to temporarily change its application icon to a fake icon, e.g., the Facebook icon shown in this demo. The risk of changing application icon is that, a skeptical user may check the application icons through the main menu to discover *Notish*. Thus, *Notish* must set an expiration time for the notification and erase it when expired.

A safer way for *Notish* to prevent exposure would be marking the spam notification as *read* and save it into the Message Box directly. As only new notifications will be displayed within the Message List, the "read" spam notifications will appear in the Message List, thus *Notish* need not change its application icon.

On the notification views of Message List, a user can tap both the messages and the application title bars. In both cases, the operations are under the control of *Notish*. The user will be navigated to the spam website in the spam notification attacks.

On the notification views of Message Box, the platform provides a set of default operations that are unrelated to the sender application. Also, the sender application can add customizable operations through the *ApplicationMenuItem*. Again, these customizable operations will delete the evidences of posted notifications and navigate the user to the spam website.

3.4.3 Attack Demonstration

We present the attack flow of both attacks in Figure 4 and 5.

Phishing Attacks: As shown in the Figure 4, by customizing the notification view, *Notish* can easily fabricate the fraudulent notifications identical to the genuine Facebook notifications. By specifying the allowed user operations on both views, *Notish* will navigate the user to a fraudulent login view of the trojan application.

One flaw of the proposed phishing attack is that *Notish* can no more cover the attack by launching the genuine Facebook application as on Android, because launching another application from a third party application is not allowed on BlackBerry OS. To reduce the suspicion, *Notish* may navigate the user to the "*password incorrect webpage*" of the genuine Facebook website(shown in Figure4).

Spam Attacks: in Figure 5, we show the demos of spam notification attacks with both Dialog and Message List. Different to the phishing notification attack, the trojan application will navigate the user to a spam website instead of a fraudulent login view.

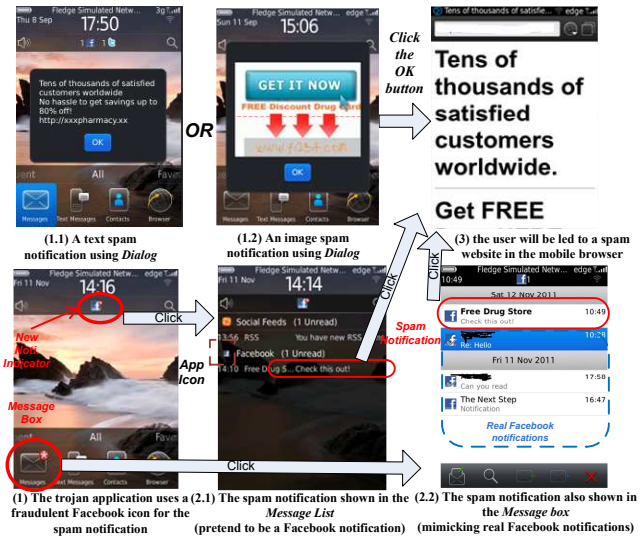


Figure 5: Dialog and Message List based spam notification attack scenarios on BlackBerry OS 7

Figure 6: A spam notification shown in the Notification Center of iOS 5 (Notish is exposed in the notification view)

3.5 Proposed Attacks on iOS

With the newly introduced Notifications Center feature, iOS 5 has still been complained by application developers for placing restrictions on notification customization [11]. Briefly, the iOS 5 provides a pop-up notification service, called *Alert*, a status bar notification service, called *Notification Center*, and an icon notification service, called badge notification.

Unlike other smartphone platforms, the user of iOS 5 decides how to display the notification on the screen. To send a notification, a third party application constructs an $UILocalNotification$ object and passes it the iOS. Depending on user preference, the notification will be posted either as an alert dialog in the center of screen or as a banner on the top of screen. No matter which form the notification is displayed, it will be saved in the *Notification Center* that can be expanded at the top of screen.

The notification view in both the Alert and Notification Center are OS-Controlled. The sender application is only allowed to provided a short text message that will be displayed in the notification view with the sender application's name and icon. We present the notification view of Alert and Notification Center in Figure 2 and Figure 6, respectively. As shown in both demos, the notification view of Alerts shows the name of the sender application, and the notification view of Notification Center displays both the name and the application icon of the sender application. Therefore, the spam notification attack fails because sending spam notification will expose the true identity of a trojan application.

Neither the phishing notification attack nor the spam notification attack is feasible on iOS 5 because of the OS-Controlled notification views. The attempting trojan application will expose its true identity on the notification view. Further, even if the user is so careless by ignoring the displayed identity of the trojan application, the allowed user operations on the alert notification is under control of the iOS instead of the trojan application. For example, when a user taps on the *View* button in an Alert or a notification in the Notification Center, iOS will navigate to the last view of the trojan

Figure 7: A spam notification on Windows Phone 7.1

application or its starting view if it has been terminated. In either case, the phishing and spam notification attack will fail because there is no way to navigate the user to the designated view.

3.6 Proposed Attacks on Windows Phone

Similar to the iOS, the phishing and spam attack on Windows Phone OS 7.1 (abs. WP 7.1) also failed because the exposure of *Notish* on the notification view. WP 7.1 provides two types of notification services. One is a pop-up notification service, called Toast, which displays notification views on the top of screen; and the other is an icon notification service, called Tile, which displays notification view on the icon of a sender application. We investigate the feasibility of sending spam notification via the Toast service.

On WP 7.1, the sender application is allowed to register a background agent, called *Scheduled Task Agent*, to the *ScheduledActionService*. This registered background agent can be invoked by system events, such as time alert, when the sender application is inactive. Within the background agent, the sender application can specify a piece of code that will be executed when the agent is invoked. In our case, the sender application can fire a *ShellToast* notification on the screen, as shown in Figure 7.

Similar to the notification center on iOS 5, the ShellToast is OS-Controlled and allows no customization on notification views. As shown in Figure 7, the application icon of the sender application (i.e., Notish in the attack) will be shown in the notification view and the sender application can not interfere it. Different from iOS 5, WP 7.1 allows the sender application to specify a property *NavigationUri* of notification, with which the user will be transferred to a designated view of the sender application when he taps on the notification view. However, the OS-Controlled notification view of ShellToast will expose the identity of a spam sending application.

4. DEFENSE MECHANISMS

Basing on the observations in the presented attacks, we propose defense mechanisms to mitigate these threats for both the smartphone platforms and the victim applications (e.g. the Facebook application). The implementation and demonstrations of proposed defense mechanisms are on Android because it is open source and is the most exploited smartphone platform.

4.1 Semi-OS-Controlled Notification

As shown in Figure 8, current smartphone platforms fall into two extremes on notification customization. On Android (left side), the most customizable smartphone platform, a third party application is given the full control over the notification service. To the opposite (on the right side), on iOS, little customization is allowed.

From the evolution of notification service, we have seen the trend of allowing customization in the notification service. However, from the proposed attacks, we see that the abuse of notification customization may enable a trojan application to launch phishing and spam notification attacks. New notification design is needed to allow notification customization while preventing the attacks.

Thus, we propose a design principle for notification view design, named *Semi-OS-Controlled* notification view, which takes into consideration both customization and notification authentication. Briefly, as presented in Figure 8(2), a Semi-OS-Controlled notification view contains an OS-Controlled frame and a App-Controlled subview. Sender authentication information, such as application icon and application name, will be displayed within the frame. The sender application cannot modify the content in the OS-Controlled frame. But, the control of subview, together with the allowed operations on this subview, can be given to the sender application.

For demonstration, we have updated the notification view from App-Controlled to Semi-OS-Controlled. Specifically, in our implementation, we modified the source code of *Toast*, *Notification*, and *NotificationManager*. Figure 9(a) presents an example when a third party application assigns a customized notification view to a status bar notification by setting *Notification.contentView*. Instead of posting the customized notification view directly, the modified *NotificationManager* will first retrieve the sender application identity from *PackageManager*, and build a frame view. The frame view wraps the received customized notification view as well as the sender application identity.

In this way, the trojan application will expose its identity with the updated notification service. Further, our modification does not affect any existing APIs thus no update is required to existing third party applications.

4.2 Notification Logging Service

On all existing smartphone platforms, a displayed notification view will be dismissed after the user operates on it. No notification logging service has been provided by the existing smartphone platforms. The Message Box on BlackBerry OS provides a simple logging service. However, the stored messages can be manipulated and even deleted by a sender application.

4.2.1 Failure of Android EventLog

Android provides an EventLog service that allows a smartphone user to access the system diagnostic event records (certain system-level events). However, by observing the event records during the proposed toast and status bar based notification attacks, we found out that the EventLog service does not include any sender application information of toast and status bar notifications.

After examining the source code of Toast, Notification, and NotificationService on Android OS, we see that, except for error messages, most logging functionalities in these source codes only work in debug mode. For example, in the NotificationService, most logging functions are called only in debug mode (i.e. when "DBG==true"); and in the Toast, all logging functions are called at a very low LOGV level. By default, these logs with LOGV are compiled out when the OS is built. Thus, in order to observe the information of sender application, the smartphone user has to rebuild the Android OS from source code with debug mode enabled (e.g., by specifying the "-eng" option when building the OS [12]). With common Android smartphones sold in stores, common users will not be able to discover the sender information by observing the event log with Android SDK.

4.2.2 Proposed Notification Logging Service

Disappointed by the default EventLog service on Android and Message Box on BlackBerry OS, we propose a Notification Logging service in the smartphone platform. On Android, we implement the Notification Logging service as a Java written system service (named *NotiLogService*) running in the Application Framework layout. *NotiLogService* starts when the smartphone is turned on and maintains a list of the latest toast and status bar notifications. The record of each piece of notification includes the triggering time and the sender application identity.

For example, as shown in Figure 8, the *NotificationManager* will add an record in the *NotiLogService* service whenever a status bar notification is triggered. As the *NotiLogService* service is a system service, installed third party applications do not have access to the service. Thus, the *NotiLogService* service itself is secured. Cur-

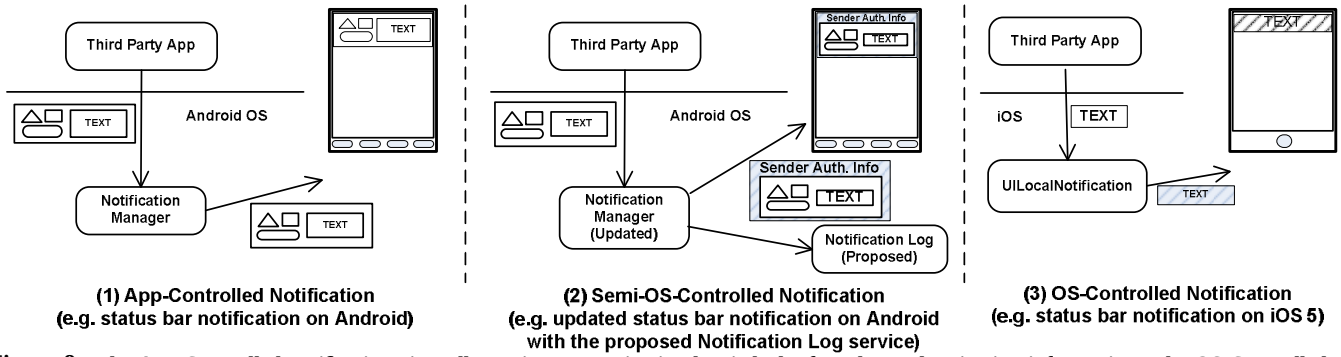
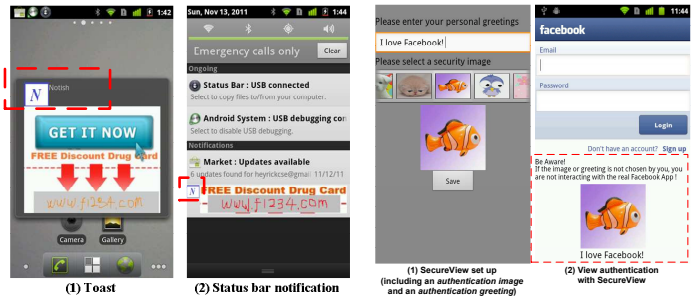


Figure 8: The App-Controlled notification view allows view customization but is lack of sender authentication information; The OS-Controlled notification view (e.g. iOS) is immune to proposed notification attacks but limits the capacity of app developers. The proposed Semi-OS-Controlled notification view supports view customization and is immune to notification attacks.



(a) Implementation of Semi-OS-Controlled notification view on Android (b) The setup and application of SecureView

Figure 9: Proposed defense mechanisms

rently, the user can access the *NotiLogService* through a dedicated UI app built with Android source code. In our future work, we are going to integrate this service into the "Settings" menu of Android.

4.3 SecureView Framework

The proposed phishing attack exposes an important vulnerability of lacking a view authentication service in the current view-based smartphone platforms. View is the basic UI of smartphone platforms. The application running in the foreground has control of the current view on the screen and accepts all user inputs on this view. However, on all current smartphone platforms, no view authentication information is displayed with the view, such as the identity of the foreground application.

In the proposed phishing attacks, the trojan application could generate fraudulent login views mimicking login views of target applications. Due to the limited UI on smartphone, it is difficult for the user to sense the difference based on the fraudulent view.

To protect the sensitive views, we propose a *SecureView* framework for a third party application to provide application authentication information on its sensitive views (e.g. login views). Specifically, using the SecureView framework, the user is allowed to choose a security image as well as writing a text-based security greeting after installing the application. Both the image and greeting are stored locally on the smartphone. This authentication information is associated with the current installation instead of the user account with this application.

Whenever a sensitive view is displayed, the application can show the security image and greeting on the sensitive view to provide view authentication to the user. In Figure 9(b), we present a demo of the setup and application of SecureView. As all applications are isolated on Android, other installed (malicious) applications cannot record the SecureView displayed on the screen.

The trust model for SecureView framework is that, there exists a secure storage for each application on the smartphone platform. This secure storage can only be accessed by its belonging application. Without subverting the platform, the trojan application cannot get knowledge about the security image and greeting, thus it cannot generate fraudulent sensitive views with valid view authentication information. This assumption is true for most existing smartphone platforms. One exception is the jail-broken iOS, in which a third party application has access to all the storage on the device.

5. DISCUSSION

5.1 Stealthiness of Trojan App

As described in Section 3, the trojan app carefully manipulates the posted notification in order to conceal its true identity from being discovered by smartphone users. Besides, other attack-specific strategies can be applied to enhance the stealthiness of attacks.

In the phishing notification attack, the trojan application mimics the target application (e.g., the Facebook application) so that the user will not realize being attacked. Another advantage for phishing notification attack is that, the number of target applications on one smartphone is very limited. To steal the user credentials of a target application, one successful attack would already be enough for the attacker. As the attack rate for phishing notification attack can be extremely low, the chance of exposing the attack and the trojan app will also be small.

In the spam notification attack, the smartphone user will certainly be aware of the unsolicited advertisements. As we described previously, the trojan application will dismiss the notification immediately after display so as to prevent being traced. On the side, one important observation is that, with the thriving of mobile app ecosystem, mobile app advertising has become popular and acceptable to smartphone users [13]. Based on this observation, two strategies may be adopted by the spam notification attacks. Firstly, the trojan app may lower the frequency of posting spam notifications. Secondly, the trojan app may take some other installed mobile applications as scapegoat. For example, the trojan application may add the spam notifications with advertisements related an installed free game so as to mislead the annoyed user.

5.2 Stealthy Spam Content Distribution

To distribute and update the spam contents on smartphones, a straightforward way is to hard code the spam content within the trojan application. The trojan application vendor may update the hard coded spam contents with the application updates. This approach is simple and requires no further networking communication between the trojan application and the trojan application vendor. However, the spam contents may be noticed by reviewers in the application review process of the application store [14]. Also, the update of spam contents can not be prompt.

A better approach for spam content distribution and update would be sending the spam content separately from the trojan application vendor to the installed trojan application. Briefly, the trojan application vendor may send a "clean" version of a trojan application to the application store first. With the "clean" version, the trojan application could bypass the application review process easily. Once the trojan application is installed, the trojan application vendor then distributes spam contents through the network communication to the installed trojan application. A spam notification update may include the text and image to be displayed in the notification view, as well as the URL of a spam website that the user will be navigated to. Besides the common Internet connections, the push services may also be abused for the spam content distribution. Due to the limit of space, we present the details of applying push notification for spam content distribution in the Appendix B.

5.3 Incentives of Proposed Attacks

In the phishing notification attacks, the attacker may make profits by selling the phished account in the underground market [15] or abusing the phished account (e.g. PayPal and bank account) directly. With the increasing number of mobile finance apps, we believe the phishing attacks on mobile apps will gain more and more attentions to the attackers.

In the spam notification attacks, the major revenue may come from the unsolicited advertisement. According to Gartner's report [13], the worldwide mobile advertising revenue was about $\$1.6$ billion in 2010 and about $\$3.3$ billion in 2011. With the proposed spam notification attacks, the trojan application can make profits from distributing unsolicited advertisements that are not accepted by normal mobile advertising company, such as AdMob.

Even when the trojan app is discovered, the loss to the trojan app developer is not significant. First of all, building a feature-rich and attractive mobile application has become easy. One quick approach to build a trojan application is to repackage existing legitimate mobile applications. As observed in [16], within the 68,187 apps the author collected from six third-party Android marketplaces, 5% to 13% apps are repackaged.

Secondly, even the trojan app is reported and removed in the official market places, there exist many third party market places and app forums [17, 18] to distribute trojan applications. Those third party markets are not that managed as the official market places, and especially attractive to smartphone users looking for free piratical mobile applications.

Thirdly, the cost of an account in the official application market places is trivial compared to the revenue gained in the proposed attacks (especially the phishing attacks). For example, on Android market place, there is only a one-time $\$25$ registration fee for a developer account.

5.4 Heuristic Approaches for Sender Application Identification

Admittedly, for security professionals, the trojan application may be identified in special environment with platform debugging tools. However, for normal users, the approaches on an off-the-shelf smartphones are limited. An intuitive approach would be examining all installed third party applications one by one. The user may uninstall a suspicious application and wait for a period of time. If no more spam notification or phishing notification is displayed, the uninstalled application would probably be the trojan application. However, this approach is neither effective nor efficient. Because the trojan application may simply lower the attack frequency or even hibernate for a period of time to decrease its suspicion. Thus, disappearance of spam or phishing notifications cannot be easily used as a direct evidence for accusing a suspicious application.

With the featured functions not related to notification services, some perseverant users may try heuristic approaches to narrow down the list of suspicious applications. We discusses several heuristic approaches that the user could use on Android and BlackBerry OS.

5.4.1 Android

On Android, there exists a delay between the time that an application requests to end itself and the time that it is actually ended by the platform. Therefore, the trojan application may still be shown in the list of running applications for a period of time even it has requested to end itself immediately after firing the notification.

Based on this observation, when the spam or phishing notification is displayed on the screen, the user may use the "*Manage Apps*" function provided by Android OS to check the list of running applications and services immediately. The "*Manage Apps*" function is very similar to the well known *Task Manager* of Windows operating systems on PC.

However, the delay is unpredictable depending on the Android OS. Further, the trojan application does not need to wait until the user sees the notification, and can stop immediately after firing the notification. The effectiveness of this heuristic approach is not guaranteed. Moreover, the list of running applications could be long, because allowing background services is a featured function on Android platform. Many third party applications keep background services, including news agent applications, online social network applications and music players.

5.4.2 BlackBerry OS

The BlackBerry OS provides a function called "*Switch Application*" that allows the user to switch among applications that are currently running. Based on our experience through experiments, we note that the BlackBerry OS stops applications faster than Android. However, when the spam or phishing notification is displayed on the screen, the user may check the listed running application. The trojan application would probably be within the list. However, the background service has been widely applied in third party applications on BlackBerry OS to keep monitoring the updates, such as news and email. Also, there is still no direct evidence to tell the trojan application from the list of running applications.

5.5 Jailbroken iOS

There exists a huge population of jailbroken iOS users [19, 20, 18]. One important feature with the jailbroken iOS is allowing customizable notification services. On jailbroken iOS 4 and iOS 5 [11], the sender application can manipulate the notification view, adding components, as well as taking control over the allowed user operations. Thus, both jail broken iOS 4 and iOS 5 are vulnerable to the proposed phishing and spam notification attacks.

In addition, the iOS 5 Notification Center on iPhone includes a weather widget and a stock widget, which are able to display customizable views in the notification center. Although no other third party widgets are available so far, WillFour20 [21] presented a demo of building customizable widgets on jailbroken iOS 5 with only open APIs. We see the possibility of allowing customizable notifications and widgets on future iOS platforms.

6. RELATED WORKS

6.1 Spam Attacks on Smartphones

Short Message Service (SMS) has been widely applied in distributing spam messages to smartphone users. To defend the SMS spam attack, the user may report the sender number to the phone carriers [22], or rely on content based filtering applications, such as the *Anti SMS Spam & Private Box* for Android [23] and *iBlackList* for iOS [24]. Cormack et al. [25] suggested that new features are needed for SMS spam filters. Vura and Venter [3] proposed an

artificial immune systems based approach to detect botnet spamming programs on Android phones. We also observed advertising Android apps, such as AirPush [26], which utilize the status bar notification to post third party advertisements on Android smartphones. Different from existing advertising apps that actively exposes their identity, our proposed trojan application hides its identity in the proposed spam notification attacks. Moreover, we show the feasibility of distribution spam notifications not only via status bar notification but also the toast notification service.

6.2 Phishing Attacks on Smartphone

Niu et al. [4] also studied the design flaws of mobile browsers that may allow web based phishing attacks. They pointed out that many featured functions of mobile browsers, such as URL truncation and hiding URL bar on page load, may cause the difficulties for user to sense the phishing webpage.

Besides Email and Browser, attempts of phishing attacks with fake applications have also been discovered. In the *09Droid* case, a programmer named "09Droid" published several fake banking applications on Google's Android Market trying to steal the user's account login information.

Also, Felt and Wagner [5] studied the feasibility of navigating the user to a fraudulent login view or login webpage by abusing the control transfer function provided on Android and iOS. One concern in their approach is the identity of trojan application is easy to reveal. For example, if the user is interacting with a malicious application before the transfer, the identity of this mobile application is known to the user. In our proposed phishing attacks, identify the trojan application will be much difficult for the smartphone user.

Recently, Schulte and Percoco [27] presented a trojan based phishing attacks on Android, which claims to be able to exploit a design flaw of Android platform so as to lead the user to a fraudulent Facebook login view controlled by an installed trojan app. Google argued that such attacks are impractical [27] . Compared to the proposed view-based phishing attacks, the notification-based phishing attacks we proposed are more practical and easier to implement. Further, demonstrations of proposed phishing notification attacks on Android are presented as well.

7. CONCLUSIONS

In this paper, we study the feasibility of launching phishing and spam attacks with an installed trojan application by abusing the customizable notification service. Experimental results and attack demonstrations are presented on four major smartphone platforms. Further, we present approaches for stealthy spam content distribution that can help the trojan applications bypass the application review process in application stores. To defend the proposed attacks, we suggest a Semi-OS-Controlled design principle for notification view, a SecureView framework for general view authentication, and a notification logging service for notification review.

8. ACKNOWLEDGMENTS

We thank the reviewers for the valuable comments. This work was supported in part by NSF CAREER 0643906. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of NSF or the U.S. Government.

9. REFERENCES

[1] C. Kanich, C. Kreibich, K. Levchenko, B. Enright, G. Voelker, V. Paxson, and S. Savage, "Spamalytics: An empirical analysis of spam marketing conversion," in *Proc. of ACM CCS'09*, 2009.

[2] R. Dhamija, J. D. Tygar, and M. Hearst, "Why phishing works," in *Proc. of the SIGCHI conference on Human Factors in computing systems*, 2006.

[3] I. Vural and H. S. Venter, "Detecting mobile spam botnets using artificial immune systems," in *IFIP Int. Conf. Digital Forensics*, 2011, pp. 183–192.

[4] Y. Niu, F. Hsu, and H. Chen, "iPhish: Phishing Vulnerabilities on Consumer Electronics," in *Proc. of UPSEC '08*, 2008.

[5] D. W. Adrienne Felt, "Phishing on mobile devices," in *Proc. of W2SP'11: WEB 2.0 Security and Privacy*, 2011.

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

[7] Wikipedia, "Mobile phone spam," http://en.wikipedia.org/wiki/Mobile_phone_spam.

[8] S. Schroeder, "2 of every 3 smartphones sold are android or ios," http://mashable.com/2011/10/06/2-of-every-3-smartphones-sold-are-android-or-ios-report/.

[9] M. Jakobsson, E. Shi, P. Golle, and R. Chow, "Implicit authentication for mobile devices," in *Proc. of HotSec Workshop '09*, 2009.

[10] G. Wright, "Facebook plist mobile security hole allows identity theft," http://garethwright.com/blog/facebook-mobile-security-hole-allows-identity-theft, 4 2012.

[11] S. Perez, "A new reason to jailbreak: Custom widgets in ios 5's notifications center," http://developersarena.com/web/2011/06/a-new-reason-to-jailbreak-custom-widgets-in-ios-5s-notifications-center/.

[12] Android, "The android open source project," http://source.android.com/.

[13] G. Inc., "Gartner says worldwide mobile advertising revenue forecast to reach 3.3 billion in 2011," http://www.gartner.com/it/page.jsp?id=1726614.

[14] Apple Inc., "App store review guidelines," https://developer.apple.com/appstore/guidelines.html.

[15] Krebsonsecurity, "How much is that phished paypal account?" http://krebsonsecurity.com/2011/10/how-much-is-that-phished-paypal-account/.

[16] W. Zhou, Y. Zhou, X. Jiang, and P. Ning, "Detecting repackaged smartphone applications in third-party android marketplaces," in *Proc. of CODASPY'12*, 2011.

[17] WeiPhone, "Weiphone forum," bbs.weiphone.com.

[18] J. Freeman, "Cydia," http://cydia.saurik.com/.

[19] J. Herrman, "ios 4 jailbroken within a day of first release," http://gizmodo.com/5558277/ios-4-jailbroken-within-a-day-of-first-release.

[20] E. Fish, "ios 5 jailbreak is already here; geeks not surprised," http://www.pcworld.com/article/241877/ios_5_jailbreak_is_already_here_geeks_not_surprised.html.

[21] WillFour20, "An example of a custom notification centre widget on ios 5," https://github.com/WillFour20/WeeAppTest.

[22] ATT, "Block spam text messages on your wireless phone," http://www.att.com/esupport/article.jsp?sid=KB115812&cv=820\#fbid=VLIsitNsUpI.

[23] D. Mate, "Anti sms spam & private box for android," https://play.google.com/store/apps/details?id=org.baole.app.antismsspam&hl=en.

[24] "iblacklist for iphone," http://www.iblacklist.com.br/.

[25] G. V. Cormack, J. M. G. Hidalgo, and E. P. Sánz, "Feature engineering for mobile (sms) spam filtering," in *Proc. of ACM SIGIR conference on Research and development in information retrieval*, ser. SIGIR '07. New York, NY, USA: ACM, 2007, pp. 871–872.

[26] airpush, "The android ad netowrk," http://www.airpush.com/.

[27] E. Mills, "Android could allow mobile ad or phishing pop-ups," http://news.cnet.com/8301-27080_3-20089123-245/android-could-allow-mobile-ad-or-phishing-pop-ups/, August 2011.

[28] G. Developers, "Android cloud to device messaging framework," http://code.google.com/android/c2dm/index.html.

[29] B. Developer, "Blackberry push service options," http://us.blackberry.com/developers/platform/pushapi.jsp.

[30] A. Inc., "Apple push notification service," http://developer.apple.com/library/mac/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/ApplePushService/ApplePushService.html.

[31] Microsoft, "Push notifications overview for windows phone," http://msdn.microsoft.com/en-us/library/ff402558(v=vs.92).aspx.

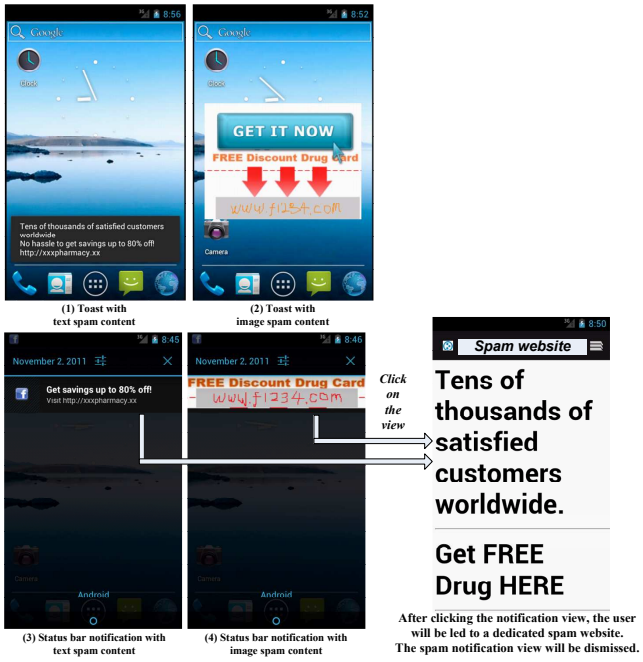


Figure 10: Demo of spam notification attack on Android 4.0

APPENDIX

A. ANDROID 4.0

Android 4.0 also provides toast and status bar notifications. We have examined the source code related to toast and status bar notification in Android 4.0.3. According to our studies, there is little change on the notification service from Android 2.3 to Android 4.0. The proposed phishing and spam notification attack will work fine on Android 4.0 with exactly the same effectiveness as that on Android 2.3. For demonstrations, we present the proposed spam notification attacks on Android 4.0 in Figure 10.

B. PUSH SERVICE

Push service is a featured service that is gaining more and more popularity. It allows third-party application servers to actively send data to their installed applications, even when the installed application is currently not running. To use the push service, the application vendor has to register at the service provider (e.g., Google and Apple). However, the push service provider has no knowledge about the contents distributed through the push service. The data distributed through the push service are messages that the vendor server wants to notify about. Depending on the handling function defined in the installed application, the pushed data may or may not be displayed directly on the screen as notifications.

Popular push services provided include the *Android Cloud to Device Messaging Framework* (C2DM) for Android [28], the *BlackBerry Push Essentials and Push Plus* for BlackBerry OS [29], the *Apple Push Notification Service* for iOS [30], and the *Microsoft Push Notification Service* for Windows Phone [31]. In the first two services on Android and BlackBerry, the pushed data are for general usage of data distribution. While, in the last two services on iOS and Windows Phone, the pushed data are specifically for notification and can be displayed directly through the local notification service. Due to the limit of space, we take the push service on Android as an example to demonstrate the application of push service in the proposed spam notification attacks.

For Android, Google provides a C2DM infrastructure that allows the third party application to push a short message to the installed application. The payload of a message is up to 1024 bytes. The smartphone receiving the message will use Intent to awake and notify the receiver application. With the received message, the notified third party application can then synchronize with the server about the details. The handling function for the received short message in C2DM is defined and controlled by the installed application, and the receiver application decides whether or not to notify the user upon the received messages.

In the proposed spam notification attacks with text spam contents, the 1024 bytes payload is already enough for the icon, text content, and the URL of spam websites. For spam notifications with image contents, the 1024 bytes restriction may limit the size of the image to display. Thus, the trojan application being notified may further synchronize with its vendor for details.

In the proposed phishing notification attacks, the C2DM service can also be applied to distribute the text and image data used in the fraudulent notification view and login view to the trojan application.