# Phishing on Mobile Devices

Adrienne Porter Felt*
University of California, Berkeley
apf@cs.berkeley.edu

David Wagner
University of California, Berkeley
daw@cs.berkeley.edu

## ABSTRACT

We assess the risk of phishing on mobile platforms. Mobile operating systems and browsers lack secure application identity indicators, so the user cannot always identify whether a link has taken her to the expected application. We conduct a systematic analysis of ways in which mobile applications and web sites link to each other. To evaluate the risk, we study 85 web sites and 100 mobile applications and discover that web sites and applications regularly ask users to type their passwords into contexts that are vulnerable to spoofing. Our implementation of sample phishing attacks on the Android and iOS platforms demonstrates that attackers can spoof legitimate applications with high accuracy, suggesting that the risk of phishing attacks on mobile platforms is greater than has previously been appreciated.

## 1. INTRODUCTION

User interfaces for mobile devices are constrained by the devices' small screens. In particular, mobile operating systems and browsers lack secure application identity indicators. A user cannot definitively tell what mobile application or web site she is interacting with. This exposes users to the risk of mistaking a malicious application for a trusted one.

Mobile applications and web sites often link to each other to share data or refer the user to a related service. For example, a music-themed web site might link the user to the iTunes application to buy a song. In a normal inter-application link, the *sender* application links to a second *target* application. After following the link, the user might provide the target application with authentication credentials or payment information.

In this paper, we discuss phishing attacks that imitate normal inter-application links. The lack of secure identity indicators means that an inter-application link could be subverted, and the user would be unable to tell that she had been sent to the wrong target. In a *direct* phishing attack, the sender is a malicious application that links the user to its own spoof screen instead of the real target application. In a *man-in-the-middle* attack, the sender is benign, but another party intercepts the link and loads a spoofed target application in place of the intended target application.

*This material is based upon work supported under a National Science Foundation Graduate Research Fellowship. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Phishing attacks exist because users become accustomed to entering their passwords in familiar, repeated settings. If users frequently encounter legitimate links whose targets prompt them for private data, then users will become conditioned to reflexively supply the requested data [9]. 40% of smartphone users enter passwords into their phones at least once a day [8]. We study 50 Android applications, 50 iOS applications, and 85 web sites to evaluate how often they link users to password-protected or payment-related applications. We find that web sites and mobile applications commonly link the user to password-protected social network and payment applications, thus conditioning users to reflexively enter their credentials after following links.

Based on this analysis of common behavior, we identify a number of new phishing attacks against mobile platforms. We demonstrate that, on Android and iOS, it is possible to build phishing attacks that convincingly mimic the types of inter-application links that our study found to be common. We categorize the attacks according to whether the sender and target are mobile applications or web sites: mobile-to-mobile, mobile-to-web, web-to-mobile, and web-to-web. For each of the four categories, we present both direct and man-in-the-middle attacks.

The contributions of this paper are:

- We believe we are the first to discuss phishing attacks between mobile applications and web sites.
- We present data on how mobile applications and web sites interact. This provides insight into the application behaviors that users are familiar with.
- We present and evaluate 15 types of phishing attacks. 13 of the attacks are novel, and 1 improves a known mobile browser spoofing technique.

## 2. BACKGROUND

We focus on Android and iOS, the two most popular modern smartphone operating systems [3]. Our phishing concerns also apply to other smartphone platforms with user interface space constraints and inter-application control transfers, such as Windows Phone 7.

### 2.1 User Interfaces on Mobile Devices

Smartphone operating systems and browsers have minimalist user interfaces to accommodate the small size of mobile devices. In particular, their user interfaces lack application identity indicators that are available in desktop operating systems and browsers.

**Mobile Application Identity.** Only one mobile application can control the screen at a time, even in platforms that support multi-tasking. The *foreground application* fills the screen and receives all user input, with the exceptions of input-method applications (e.g., alternate keyboards) and the Apple App Store (which uses pop-up dialogs for in-application purchases). No major smartphone operating system displays the identity of the foreground application. Users can find a list of running processes, but no part of the screen identifies the current foreground application.

**Web Site Identity.** Mobile browsers display only one browser window at a time. The Android and iOS browsers display the URL of the current window at the top of the screen, similar to desktop browsers. However, web sites can hide the URL bar once the page has loaded; this commonly used feature allows the web site to fill the available screen. (For example, Figure 2a shows `weather.com` in mobile Safari without the URL bar.) If a user wants to see the URL bar, she can tap the top of the screen.

## 2.2 Platform Security

Mobile applications are less trusted than their desktop counterparts. In Android and iOS, applications are isolated from each other by default; for example, applications cannot read each other's databases or network traffic. Additionally, Google and Apple attempt to prevent users from installing malware by controlling how users install applications.

Android applications are restricted with an application permission system. Permissions control access to privacy- and security-relevant parts of the system API, such as the ability to read a user's contacts or send a text message. When a user installs an application, the application's desired permissions are displayed to the user. The user can then decide whether to grant all of the permissions and continue with the installation. This informs users of the risks of installing an application.

Users of iOS devices can only install applications from the App Store. In order to be listed in the App Store, applications must undergo a formal review process. Although specific details of the App Store review process are secret, the process likely includes a security review.

Web sites in mobile browsers are also treated as potentially untrustworthy. They must obey the standard Same Origin Policy [15], meaning that web sites on different domains are isolated from each other. Neither Android nor Apple review or restrict access to mobile web sites.

## 2.3 Control Transfers

A *control transfer* occurs when the foreground application or web site changes. As a result of a control transfer, the user stops interacting with one application and begins interacting with another. Although mobile applications and web sites are isolated, inter-application communication via explicit channels can lead to control transfers. Control transfers can happen in four ways, classified according to whether the sender and target are mobile applications or web sites:

- *Mobile Sender ⇒ Mobile Target.* Android and iOS applications can optionally choose to accept communication from other applications by registering a receiver with the operating system. In response to a message, the OS transfers control to the target application.

- *Mobile Sender ⇒ Web Target.* A mobile application can send the user to a web site by invoking the phone's native browser with the target URL. The browser will come to the foreground with the target web site.

  Alternately, a mobile application can *embed* web content. As a result, the web content is rendered as part of the mobile application's screen. For example, Figure 1b shows Groupon embedding a Twitter login page. Embedding can be done by communicating directly with a web site's data API or by asking the OS to load a web site in a WebView. A *WebView* is similar to an `iframe`: the embedding application can control the size and placement of it within the screen, and browser chrome is not displayed (including the address bar). Mobile applications can additionally insert arbitrary JavaScript into a WebView. The two types of embedded web content are difficult to visually differentiate. For both, the mobile application retains control of user input and the screen while the user interacts with the embedded web content.

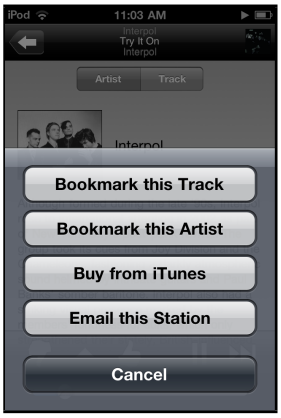
- *Web Sender ⇒ Mobile Target.* A web site can send the user to a mobile application if the mobile application registers a data scheme with the OS. A *data scheme* is a non-standard URL protocol. For example, the mobile Skype application registers the `skype` scheme. Thereafter, clicking on a `skype://` link in any web site will send the user into the Skype application if it is installed. While this example involves explicit user action, user involvement is not necessary for such a transfer; a web site can automatically launch an application by embedding an `iframe` with the appropriate scheme. (The target application will take full control of the UI, despite the use of an `iframe`.)

  Applications can also register for HTTP(S) domains. For example, the mobile Google Maps application registers `maps.google.com`. When a web site links to an application-registered domain in Android, the user is asked to choose between the browser and the alternate application. After choosing the first time for a domain, the user can set a default application. In iOS, only a small number of Apple-approved applications (like the App Store and YouTube) can register themselves for HTTP(S) domains, and the registered applications always override the browser and receive control.

- *Web Sender ⇒ Web Target.* Web-to-web control transfers are standard links. As on the desktop, web sites can link to each other, automatically redirect, and embed each other in `iframes`. The primary difference in a mobile setting is that either the target or sender can hide the URL bar.

A developer may not be responsible for a link, e.g., a link in an e-mail. We focus on control transfers that developers have intentionally inserted into their applications.

Mobile applications (with the exception of financial applications) commonly store their users' passwords so that users won't need to re-enter them. However, users must enter their passwords into the web versions of these applications, either in the browser or as embedded web content. Thus, users are still conditioned to enter passwords when they see an appropriate login screen for a mobile target.




(a) Pandora's song description screen includes this menu. One button opens iTunes to the song, and another opens the Mail application with a pre-composed e-mail. If the user decides to purchase the song, she will be asked for her iTunes password.


(b) Groupon invites users to share daily deals with friends. The Twitter button takes the user to this screen, which is still part of Groupon. Groupon created the top "Twitter Connect" bar, and the Twitter login is embedded below.

Figure 1: Screenshots of Pandora and Groupon on iOS.

# 3. COMMON CONTROL TRANSFERS

As users interact with legitimate applications, they become conditioned to enter their passwords and payment information in certain types of settings [9]. Phishing attacks prey on user expectations by mimicking and then subverting legitimate application behavior. We study popular Android, iOS, and web applications to identify how often and when control transfers lead to password or payment entry. Asking a user to enter a password after a control transfer raises phishing concerns because the user may not know the correct identity of the post-transfer application.

## 3.1 Mobile Sender

We studied the control transfer patterns of the 50 most popular free Android 2.2[1] and iOS 4.3[2] applications. We manually exercised each application and recorded whether it sends the user to other mobile applications, opens web sites in the browser, or embeds web content. Manual testing provides a lower bound on the number of control transfers. We also observed whether each control transfer involves the entry of information pertaining to:

- *Passwords.* Is the target application's content only visible to logged-in users? (For example, a user must log in to Facebook to post to her Facebook "wall".) If so, this indicates that the user will not be surprised if the target application requests a password.
- *Payment.* Does the sender link to the target for the explicit purpose of payment? (For example, Amazon sells books and music.) If so, the user is prepared to enter payment information into the target application.

We found that 89 of the 100 mobile applications contain links that send the user to another application or web site.

[1]The Android applications were collected in October 2010.
[2]The iOS applications were collected in March 2011. We tested iPhone/iPod Touch applications.

*Mobile Sender, Mobile Target*

| *Mobile Target* | *Android* | *iOS* |
|---|---|---|
| Another mobile application | 56% | 72% |
| A password-protected application | 36% | 60% |
| An application for payment | 10% | 34% |

Table 1: The rate at which 50 Android and 50 iOS applications link the user to another mobile application. Targets may be both password-protected and payment-related. We do not include the browser as a target mobile application; the browser is considered separately in Table 2.

### 3.1.1 *Mobile Sender ⇒ Mobile Target*

Table 1 summarizes the results of our mobile-to-mobile study. A majority of the studied mobile applications contain functionality that will send the user to other mobile applications. Figure 1a provides an example of a mobile-to-mobile control transfer. We observed four common types of control transfers that involve passwords or payment: social sharing and the purchase of upgrades, music, and credits.

**Sharing.** Some applications encourage users to share content (e.g., high scores) with their Facebook, Twitter, or e-mail contacts. This involves transferring the user to a Facebook, Twitter, or e-mail application. These target applications are password-protected.

**Upgrades.** Developers sometimes publish two versions of an application: a free version with limited functionality or banner advertisements, and a paid version with full functionality or no advertisements. The free versions of these applications link to the Android Market or Apple App Store to "Upgrade" or "Remove ads." In order to complete the upgrade, the user may need to enter password or payment information into the store. The Android Market asks the user for payment information for paid upgrades, and the Apple App Store prompts the user for a password for both free and paid downloads. The Apple App Store also frequently asks the user to verify stored payment information. Consequently, users will likely be accustomed to entering their password after control transfers to an application store.

All of the applications in our sample set were free at the time of download; users of paid applications will see fewer payment links than our results suggest. However, the most popular free applications are downloaded orders of magnitude more often than the most popular paid applications.

**Music.** Music-centric applications typically contain links to purchase songs. In iOS, these links point to the Apple iTunes Store. For Android, some devices come with a pre-installed Amazon MP3 application for this purpose. Both target applications may require passwords or payment.

**Credits.** Our sample of popular iOS applications includes 34 games, many of which sell game credits through the Apple Game Center or App Store. Game credits are required to progress through the games. The Game Center and App Store both immediately request a password to complete the sale. Periodically, they will also ask the user to verify her payment information. We did not observe the sale of game credits in our sample of Android applications. However, our sample includes only 1 game, so our analysis may undercount the rate at which game applications can transfer control to a payment application.

*Mobile Sender, Web Target*

| *Embedded Web Target* | *Android* | *iOS* |
|---|---|---|
| Another company's website | 16% | 46% |
| A password-protected web site | 8% | 38% |
| A web site for payment | 2% | 0% |

| *Web Browser Target* | *Android* | *iOS* |
|---|---|---|
| A web site | 30% | 18% |
| A password-protected web site | 3% | 4% |
| A web site for payment | 2% | 0% |

Table 2: The rates at which 50 Android and 50 iOS applications embed web sites or open web sites in the browser. Targets may be both password-protected and payment-related. We suspect that Android web payment rates are low because of biases introduced by sampling only the most popular applications.

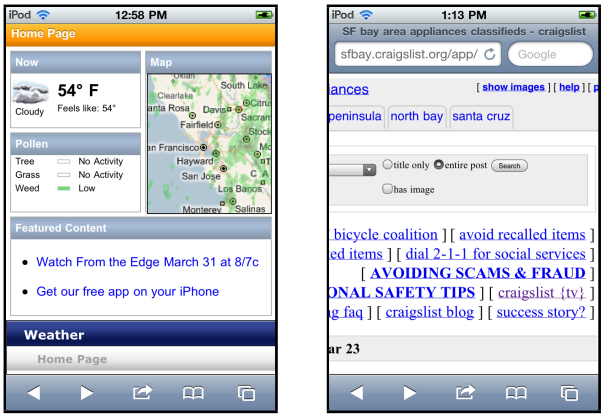
Our study demonstrates that users of mobile applications are accustomed to seeing links to other mobile applications for social sharing, upgrades, music, and credits. Links to e-mail, Facebook, Twitter, and the platform stores are particularly prevalent. Users are conditioned to enter their passwords for these operations. Phishing attacks could therefore mimic these control transfers and prompt for a password without deviating from user expectations.

### 3.1.2 *Mobile Sender ⇒ Web Target*

Mobile applications can display web content by sending the user to the browser or embedding the content. Table 2 summarizes how often the sampled applications use either approach, and Figure 1b displays an example from our sample set. For embedded web content, we only report the inclusion of web content from other companies; many mobile applications seamlessly embed their own web content, but that practice is not relevant to phishing.

**Passwords.** Social sharing accounts for nearly all of the links from mobile applications to password-protected web sites. For iOS applications, social sharing is primarily implemented with embedded web content. Within our Android sample set, applications do not commonly link to password-protected web sites. Instead, they prefer to implement social sharing with links to mobile applications. This is because many Android devices come with pre-installed Facebook and Twitter mobile application clients.

**Payment.** Within our sample set, web sites are not commonly used for payment. The iOS applications rely on Apple-provided applications (e.g., iTunes and the App Store) to handle financial transactions; this obviates the need for web-based payments. Two Android applications refer users to web sites for purchases, but most of the Android applications that we studied either process their own payments within the application or rely on the Android Market. We suspect that this is an artifact of popularity; many of the very most popular Android applications are developed by large companies with their own payment processing services. We have independently observed other, less popular Android applications using PayPal for donations and non-Market upgrades. For example, PayPal's web site showcases several PayPal-enabled applications with a few hundred to a few hundred thousand users [11]. If our sample had included less popular applications, we expect we would have observed a higher rate of web payments among Android applications.




(a) The Weather Channel links the user to the App Store.

(b) Craigslist links the user to YouTube for Craigslist TV.

Figure 2: Web sites with links to mobile applications, in iOS.

Our analysis indicates that users who participate in social sharing are at a high risk for falling for a phishing attack. 38% of the sample iOS applications embed password-protected web content, and all of them do so for social sharing. Furthermore, a user must log in to a web site every time she encounters it in a new application because there is no cross-application cookie store for embedded web content. This means that social users may enter their social networking site passwords into more than a third of the applications they install, thereby acclimating them to this workflow.

On the other hand, we find that popular mobile applications do not commonly link users to web sites for payment. This reduces the likelihood that people are conditioned to enter their passwords or payment information into a phishing application that subverts the web-to-mobile scenario.

## 3.2 Web Sender

We studied 85 of the 100 most popular Alexa-ranked domains[3] to evaluate web site link behavior. For each domain, we collected the links on the home page and one representative page of content, as linked off the home page. We used iPhone and Nexus One browser user-agent strings to receive the appropriate mobile versions of the web sites.

In order to collect the links, we built a Firefox extension to crawl each loaded page and find link target values. Some web sites use JavaScript to create links in a non-standard way; as a heuristic for finding these cases, the extension also collected `onclick` values for `div`, `span`, and `img` elements. We then reviewed the `onclick` JavaScript to find links, but we did not follow method calls referenced in the `onclick` event handler. We may have missed complex `onclick` links or links that are dynamically generated after load.

### 3.2.1 *Web Sender ⇒ Mobile Target*

A link to a mobile application could fail, if the application is not installed. However, certain applications are guaranteed to be present on every platform. iOS and Android phones have common applications like Google Maps, YouTube, and an e-mail client. Each operating system also has its own applications (for example, iOS includes the iTunes

[3]We omitted adult and advertising web sites from the list of 100. Advertising home pages do not reflect the content that users typically see from that domain.

*Web Sender, Mobile Target*

| *Core Application Target* | *Android* | *iOS* |
|---|---|---|
| A core mobile application | 38% | 47% |
| A password-protected application | 22% | 41% |
| An application for payment | 6% | 25% |

| *Total Application Target* | *Android* | *iOS* |
|---|---|---|
| A mobile application | 49% | 48% |
| A password-protected application | 38% | 42% |
| An application for payment | 6% | 25% |

Table 3: The rates at which 85 web sites include links to mobile applications. The top counts only links to core applications (which are present on every phone). The bottom counts all links to any application, core or not. Targets may be categorized as both password-protected and payment-related.

store). We refer to these applications as *core* applications. Links to core applications are guaranteed to succeed.

Table 3 presents the rates at which web sites link to the core applications. The Android and iOS rates differ partly because of differences in their sets of core applications. Table 3 also provides the total link rate, which includes links to both core and non-core applications. The total link rate includes application-defined schemes (like `hulu`). For Android we also count links to `http(s)` domains that have been registered by the 100 most popular Android applications.

**Passwords.** Web sites link to mobile applications for the same reasons as mobile applications link to other mobile applications. Many web sites contain `mailto` or `twitter.com` links to share content with friends; `mailto` links open the mobile e-mail client (a core application), and the `twitter.com` domain is registered by a popular and often pre-installed Android application. `mailto` links are also sometimes used to contact the web site staff.

**Payment.** Some web sites link to the Apple App Store or Android Market to let the user download their application or buy related items. (In fact, some web sites such as Hulu are not fully functional on mobile browsers, so the user must install the application to use the service.) The user may need to enter his or her account password or verify payment information to install the given application.

Our web analysis shows that web sites commonly link to mobile e-mail and Twitter applications. Twitter, in particular, is an attractive phishing target. We also found that web sites often link users to the Apple App Store or Android Market to install the company's mobile application, which indicates that the web-to-mobile installation process could become a target for phishing attacks.

### 3.2.2 *Web Sender ⇒ Web Target*

Web-to-web links are a standard part of the Internet. All but one of the web sites we crawled contain multiple links to external domains. Although we did not measure their use in our data set, external payment services like PayPal and Google Checkout are widely incorporated into web sites.

# 4. PHISHING ATTACKS

We discuss how phishing attacks can be mounted against each of the four control transfer scenarios enumerated in Section 2.3. For each scenario, we present two types of attacks: direct attacks and man-in-the-middle attacks. In a *direct* attack, the sender application is malicious and loads a fraudulent target application. In a *man-in-the-middle* attack, the sender and target applications are both benign, but a malicious party intercepts the control transfer and responds in place of the legitimate target.

The goal of our attacker is to mimic the legitimate application behavior that we identified in the application survey (Section 3). An accurate attack should not deviate from the user's expected workflow, and the fake user interface should be indistinguishable from the target user interface. The user should have few or no opportunities to differentiate the phishing attack from legitimate behavior. We evaluate how well each attack meets these accuracy goals.

## 4.1 Mobile Sender ⇒ Mobile Target

In this scenario, the user believes that one mobile application links to another, trusted application. In addition to mimicking a normal workflow and user interface, malicious mobile applications must face the Android permission model and Apple review process.

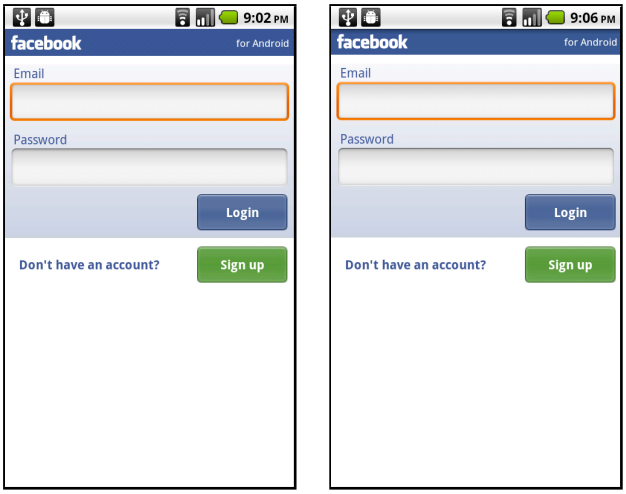
### 4.1.1 *Direct Attack*

As presented in Section 3.1.1, mobile applications commonly include social sharing and payment buttons. A malicious application could similarly include "Share on Facebook" or "Upgrade this application" buttons. Clicking on one of the buttons would send the user to a screen that spoofs the target application. The phishing screen could request the user's password or payment credentials, enabling the malicious application to steal the data. The phishing application would then load the real application. If the user does not have an existing session with the real application, then the real application will ask the user to enter her password. This resembles normal application behavior after a failed login attempt, so the user might naturally assume that she had mistyped her password.

*Evaluation.* Mobile login screens are often very simple, which makes them easy to copy. Figure 3 shows a fake Facebook login screen, which we constructed in several hours using images and layout values copied from a disassembled version of the legitimate Android Facebook application. It is highly unlikely that any user could differentiate between the real Facebook login screen and our fake Facebook login screen.

Android's permission system would do little to warn users of this attack; the attack requires no permissions, which might give users a false sense of security. At most, the malicious application might request the `INTERNET` permission to send the stolen data to the attacker. The permission request would not be anomalous: 87% of free applications request Internet access [6]. However, the `INTERNET` permission is not required because the Android API provides several ways to submit web requests and exfiltrate captured data without the `INTERNET` permission. For example, a MediaPlayer object can be created to load an arbitrary HTTP URL [2].

The Apple application review process might prevent this attack from appearing in the App Store if reviewers detect the fraudulent screen. However, the review process is not perfect [14] and there is no guarantee that reviewers would detect such an attack. More dangerously, the attacker could use web content to evade detection during review. The fraudulent screen would be constructed with an embedded web site that is the full size of the screen, served by the attacker. (Recall that embedded web sites do not have browser chrome.) During the application review, the web site could




(a) Real Facebook login. (b) Fake Facebook login.

Figure 3: Our Facebook phishing application for Android. When a user enters her password, the credentials are captured before invoking the real Facebook application. The other buttons (e.g., "Sign up") lead directly to the Facebook's corresponding page.

immediately redirect the user to the legitimate application with no signs of any malice. Once the application has been added to the store, or at any subsequent date of the attacker's choice, the attacker could change the web site to a fraudulent copy of the target. As discussed further in Section 4.3.1, web content can be styled to mimic the look and feel of an application. Alternately, the attacker could try to evade detection by targeting only a subset of the user population or not serving the attack to users in a certain geographic region (e.g., Apple's headquarters in Cupertino).

### 4.1.2 *Man-In-The-Middle*

Man-in-the-middle attacks can be launched on mobile applications in two ways. The first attack, scheme squatting, is weak because it changes the user's workflow and cannot be hidden from application reviewers. Task interception is a strong attack, but it can only be implemented on Android.

**Scheme Squatting.** Some applications register to handle schemes. If the the real application for a scheme is not installed, a malicious application could register for the scheme instead. Messages intended for the target would instead be delivered to the attacker, and the attacker could present the user with a phishing screen upon launching. Scheme squatting is a form of an attack named "Activity hijacking" [4].

*Evaluation.* One strength of this attack is that it does not require any Android permissions. However, it suffers from three weaknesses. First, to evade detection, the application would need to emulate the intended target after the login screen completes. Otherwise, the user's expected operation would abruptly halt, raising suspicion. Emulation may be implausible, and this problem is intrinsic to the attack. Second, the real application might be installed. If so, iOS and Android would give the user a choice between the real application or the phishing application. Consequently, the phishing application would need to convince the user to prefer it to the real application, which would likely fail. Third, it would not be possible to hide this behavior from reviewers because applications must declare their schemes upfront in a file that is bundled with the application.

**Task Interception.** If an OS lets applications view the list of running processes, then a malicious application could poll the task list and wait for a target application to become active. The attacker could then launch itself and display a phishing screen when it is brought to the foreground. When the user enters credentials into the phishing screen, the fraudulent application can exit, leaving the original target application visible. We implemented this attack for Android and found that the application needs to poll the running task list every 5ms to prevent the user from noticing that a new screen replaces the original target. At 5ms or below, the transition is not detectable to the naked eye.

*Evaluation.* Task interception is a very effective attack for Android. The attack requires two permissions: one to start the application in the background after the phone has booted, and another to request the task list. However, both permissions are considered non-dangerous, and neither is displayed to the user during installation. We are not aware of a way to launch this attack on iOS because applications are not allowed to view the task list.

## 4.2 Mobile Sender ⇒ Web Target

In this scenario, the user believes that a mobile application is displaying or linking to a trusted web site.

### 4.2.1 *Direct Attack*

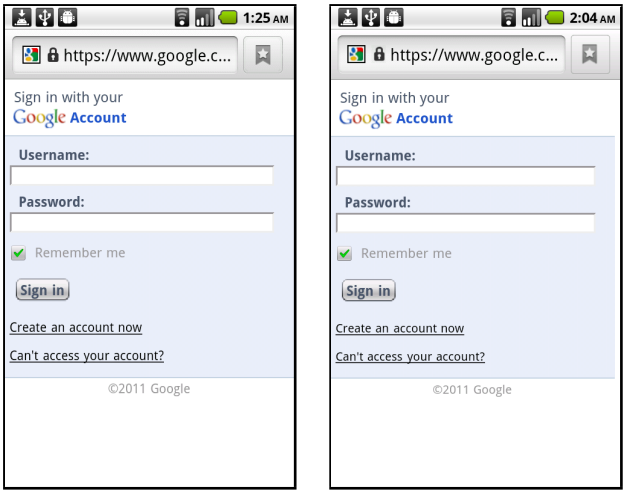
A malicious application could present the user with buttons to share content or purchase an item. The attacker can then eavesdrop on the user's interaction with an embedded web site or present the user with a site in a fake browser.

**Embedded Web Content.** Many iOS applications embed pages from password-protected sites, such as Facebook and Twitter, into their screens. This trains users to type their password wherever they see a password prompt, despite the lack of security indicators. The risk is obvious: a malicious application can embed a web site and then eavesdrop on any exchanged credentials. Both Android and iOS allow applications to insert JavaScript into embedded WebViews, and there is no mechanism for web sites to prevent this.

*Evaluation.* This attack is close to indistinguishable from legitimate behavior. The inserted JavaScript could be obfuscated to thwart application analysis. The Android version of this attack would require the `INTERNET` permission to load the target, but so would the legitimate version.

**Web Browser.** If the target web site hides its URL bar, then the mobile application could load a phishing web site in the browser. The phishing web site would then hide the URL bar. Many password-protected web sites (such as Bank of America, Amazon, and Facebook) hide their URL bars [10].

If the target web site does not hide its URL bar, then the attacker can spoof the trusted browser chrome. There are two ways to do this. First, the attacker could launch the browser, hide the URL bar, and then present a fake URL bar. We discuss how to build spoofed URL bars in Section 4.4.1. Alternately, the attacker could display a fake browser that looks identical to the real browser, except it lies about the current URL. Unlike their desktop counterparts, mobile browser interfaces are not customizable; this makes them easier to falsify. The fake browser could load the real




(a) Real browser. (b) Our fake browser.

Figure 4: Our fake Android browser for a Nexus One phone. It displays the legitimate GMail website.

web site in a WebView and listen to the user's keystrokes. If the user were to try to use features of the browser that the fake browser can't implement (e.g., history), then the attacker needs to end the attack and launch the real browser. We implemented this for Android (see Figure 4).

*Evaluation.* The first attack is extremely simple and could be effective with users who do not check browser chrome for security indicators. It does not require any notable Android permissions, and the target web site could be benign until after Apple's review process. A security-conscious user could detect the ongoing attack by deciding to check the URL bar, although a user study shows that users do not do this in practice [10]. The second attack is more complete, although the user might notice that the bookmarks button next to the URL bar is not functional. Niu et al. previously demonstrated that spoofed URL bars can be convincing enough to trick security experts [10]. The third attack would likely fool users despite some shortcomings. Real browsers have data and functionality that a fake browser cannot replicate (e.g., history). For example, the Nexus One browser has 14 menu items, 3 of which our fake browser cannot handle smoothly. However, it is unlikely that a user would press these buttons before entering her password.

### 4.2.2 *Man-In-The-Middle*

When users of mobile devices connect to the Internet over insecure WiFi hotspots, they are at risk of man-in-the-middle attacks. For instance, in one well-known active attack [12], if a user navigates to an HTTP web site with an HTTPS login form, then the network attacker can change the HTTP web content so that the login form submits the password to the attacker's server. Security-conscious web sites defend against this attack by not including login forms in HTTP pages. Instead, the user must click on a link to go to a separate, all-HTTPS login page. If the attacker redirects the user to a login page on a different domain, the user has the opportunity to notice that the login page in the URL bar is from the wrong domain. This defense against network attacks relies on the presence of a trusted URL bar, which may not be present on a phone.

**Embedded Web Content.** Consider a legitimate mobile application that embeds a web page served over HTTP. A network attacker could change the "login" button on this page so that it links to a page owned by the attacker. When the user clicks this button, she will be taken to a phishing page within the embedded web frame. Since there is no URL bar for embedded content, there is no way for the user to detect that she has left the original web site. The attacker can thus steal the user's credentials. To better mimic the user's expected workflow, the attacker could then relay the credentials to the valid web site and sign the user in.

*Evaluation.* This attack is not detectable by the user. Of our 100 sampled applications, 4 applications embed HTTP content with login links.

**Web Browser.** A similar attack is possible when a legitimate mobile application links the user to an HTTP web page that will be rendered in the browser. Normally, when not under attack, a browser would display a URL bar that indicates what site the user is currently browsing. However, this does not pose a barrier to the attack because mobile browser chrome can be hidden and spoofed. (Section 4.4.1 describes how to spoof mobile browser chrome.)

*Evaluation.* This attack tricks the user with a spoofed URL bar, which can be made almost indistinguishable from real browser chrome. Of our 100 sampled applications, 7 link to HTTP content with login links in the browser. For example, the popular Android application Shazam links to songs on `http://www.amazon.com` for the user to purchase them.

Applications can prevent these attacks by only sending users to HTTPS web pages, and never to a HTTP web page. The site must support HTTPS for all of its pages.

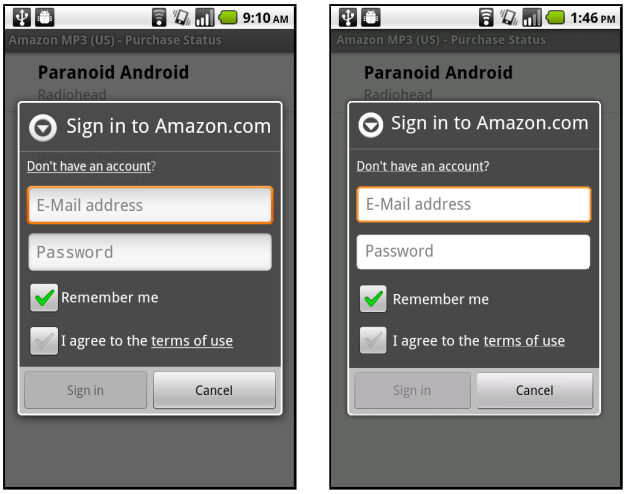
## 4.3 Web Sender ⇒ Mobile Target

In this scenario, the user interacts with a web site in the browser. The web site links to a trusted mobile application.

### 4.3.1 *Direct Attack*

Web sites often link the user to mobile applications, as discussed in Section 3.2.1. Malicious web sites could mimic this behavior without appearing suspicious. For example, a phishing site might provide a link to buy a song from the iTunes Store. When the user clicks on the button, the web site can pretend to transfer the user to the target application, but actually display a fake version of the target application. The malicious web site must hide the URL bar and style the web site to respond to user touch in the same way as the mobile application. For example, in Android, form fields should have rounded edges and turn orange when selected. Title text and dialog boxes need subtle drop shadows, and the browser's zoom feature should be disabled unless the application also supports zooming.

*Evaluation.* As we demonstrate in Figure 5, it is possible to build a web site that looks very similar to a mobile application. However, the degree to which a web site can emulate an application differs by platform. In iOS, the mobile Safari browser places a navigation bar on the bottom of the screen. This makes it impossible for a normal web site to occupy the entire screen. However, users can "install" web sites to their home page (i.e., add an icon that launches the browser to that page). Installed web sites can remove the navigation button to mount an attack that would be close to unde-




(a) Real application. (b) Web site in the browser, spoofing the application.

Figure 5: Our web site spoofs the Amazon MP3 Store (an Android application). The browser is the default Nexus One browser.

tectable. However, it may not be feasible for an attacker to convince a user to install a web site.

In Android, one unavoidable difference is how the spoofed web application responds to the physical Menu button on the phone. A web site cannot change the options that the browser presents, and the browser will have different Menu options than the target application. However, few users will try to use the Menu when presented with a login screen that does not allow any other actions. The user will likely be focused on the task of logging in.

#### 4.3.2 *Man-In-The-Middle*

Web-to-mobile links are vulnerable to the same man-in-the-middle attacks as mobile-to-mobile links (Section 4.1.2), with the same strengths and weaknesses. A malicious mobile application can scheme squat or watch the task list for the start of a target application.

### 4.4 Web Sender ⇒ Web Target

In this scenario, the user interacts with a web site in the browser. The web site links the user to a trusted web site.

#### 4.4.1 *Direct Attack*

Niu et al. present detailed web-to-web phishing attacks on mobile devices [10]. They discuss how phishing attacks can be built by hiding the browser URL bar or spoofing the URL bar. URL bar spoofing has three steps: (1) the attacker hides the legitimate URL bar, (2) the attacker adds a spoofed URL bar to the visible top of the page, and (3) the attacker catches any attempts to scroll to the real top of the page and instead jumps to the fake URL bar [10, 13].

Niu et al. describe several differences between the behavior of their spoofed URL bar and the real browser user interface. We find that we can reduce the obviousness of two of these glitches. First, they always send the user to the fake top of the page whenever the user attempts to scroll in any direction. The users in their user study find this annoying. We ease the intrusiveness of our scrolling control by always scrolling a fixed amount away from the current location in the document, but not to a specific location. The scroll event therefore might scroll slightly more or less than the user intended, but the screen will never unexpectedly shift a large amount. Second, they observe that the URL bar is visible while the page loads and renders, giving the user the opportunity to see the real URL. We reduce the URL loading and rendering time by delivering a phishing page that is empty except for a script. The page load will complete very quickly, so the user cannot read the real URL bar. After the page load event completes, the script dynamically adds all of the visible elements to the phishing web site.

*Evaluation.* An observant user could unmask the URL bar hiding attack by scrolling up to view the URL. There are also two weaknesses in the spoofing attack. First, the bookmarks button is placed next to the URL bar. In iOS, the bookmarks cannot be faked, and the user might notice a non-functional bookmarks button. (In Android, an application with the appropriate bookmark permissions can access the browser's bookmark storage.) Second, both browsers have a window selection feature that lets the user view the URLs of the open windows.

Despite these shortcomings, Niu et al. found that the URL bar hiding and spoofing attacks were very successful on a set of 37 users that included security experts. None of the users identified the phishing attacks when the URL bar was hidden, and only 1 noticed the fake URL bar. The user who noticed the fake URL bar did so because of a small implementation error in their emulation of the browser chrome. The near-perfect success rate of their phishing attacks on primed, expert users indicates that these techniques have the potential to enable highly successful phishing attacks. Their user study found that small glitches (e.g., a non-functional bookmark button) were attributed to browser or web site bugs and did not raise suspicion.

#### 4.4.2 *Man-In-The-Middle*

In Section 4.2.2, we describe an active attacker that subverts HTTPS form submission after a mobile application has linked the user to an HTTP site. An even more powerful network attack is possible in the web-to-web setting. If the user ever visits any HTTP page in the browser while there is an active attacker tampering with the network connection, then all subsequent browsing can be compromised. A network attacker could subvert all HTTP web pages so that all links to HTTPS pages point to an attacker-controlled server. In a desktop browser, the user would have an opportunity to detect the attack by looking at the URL bar. However, in a mobile browser, the attacker can suppress visual indicators of the attack with a fake URL bar. This attack could be automated by a tool like Ettercap [1].

*Evaluation.* This attack is unobtrusive and would likely go undetected by users. It relies on the mobile browser URL hiding and spoofing techniques, which are demonstrably effective at tricking users. The only way for the user to avoid the attack is to manually enter HTTPS URLs directly into the browser URL bar (or with a saved bookmark).

## 5. RISK EVALUATION

We consider how likely different phishing attacks are, given the prevalence of the legitimate application behaviors. We also discuss the most targeted web sites.

| Legitimate Behavior | Prevalence | Attack Technique | Accuracy |
|---|---|---|---|
| *Mobile Sender → Mobile Target* | | | |
| Social sharing, upgrades, game credits | Very Common | Fake mobile login screen | Perfect |
| Social sharing, upgrades, game credits | Very Common | Task interception | Perfect |
| Social sharing, upgrades, game credits | Very Common | Scheme squatting | Low |
| *Mobile Sender → Web Target* | | | |
| Embedded login pages | Common | Keylogging | Perfect |
| Opening a target in the browser | Very Uncommon | URL bar hiding | High |
| Opening a target in the browser | Very Uncommon | URL bar Spoofing | High |
| Opening a target in the browser | Very Uncommon | Fake browser | High |
| Embedded HTTP page links to HTTPS login | Very Uncommon | Active network attack | Perfect |
| App sends user to HTTP page in browser that links to HTTPS login | Uncommon | Active network attack + URL bar spoofing | High |
| *Web Sender → Mobile Target* | | | |
| Link to mobile e-mail or Twitter | Common | Web site spoofs mobile app | High |
| Link to mobile e-mail or Twitter | Common | Task interception | Perfect |
| Link to mobile e-mail or Twitter | Common | Scheme squatting | Low |
| *Web Sender → Web Target* | | | |
| Payment via PayPal or Google Checkout | Common | Hide the URL bar | High |
| Payment via PayPal or Google Checkout | Common | Spoof the URL bar | High |
| User follows link from HTTP to HTTPS | Very Common | Active network attack + URL bar spoofing | High |

Table 4: We match each attack technique with the legitimate behavior that it subverts, along with how common the legitimate behavior is. The most effective attacks mimic common behavior with perfect accuracy. "Perfect" accuracy means the user cannot distinguish the attack from the original, and "high" accuracy means the user can only identify the attack by doing something unusual.

## 5.1 Attacks

A phishing attack is comprised of two components: the legitimate behavior that it mimics, and the technique used to carry out the attack. An attack's plausibility is therefore a combination of (1) how *prevalent* the legitimate behavior is in real applications, and (2) how *accurately* the attacker can copy the legitimate behavior. Table 4 summarizes all of the phishing attacks discussed in this paper, along with the prevalence of their corresponding behaviors and an evaluation of the accuracy of the mimicry. We judge prevalence using the data from our study of iOS applications, Android applications, and mobile web sites (Section 3). We assign an accuracy score based on our evaluation of the strengths and weaknesses of each attack technique (Section 4).

The most effective attacks mimic common behavior with techniques that perfectly match the target workflow. Of the 15 attacks presented in this paper, 8 mimic common legitimate behaviors and use highly accurate attack techniques. 5 more use highly effective attack techniques, but their associated behaviors are uncommon within our set of applications.

## 5.2 Targets

We identify the password-protected applications that are linked to the most often by the applications in our data set. These target applications are at the highest risk of phishing attacks because legitimate links to them are so common.

Facebook and Twitter are the most common legitimate link targets. This is likely because developers want to encourage sharing on social networks for the free marketing. The 100 iOS and Android applications link to Facebook 35 times and Twitter 20 times. Notably, login forms for Facebook and Twitter were embedded in 19 and 12 applications, respectively. Additionally, 13 of the 85 web sites link to Twitter. This indicates that Facebook and Twitter would be ideal candidates for phishing attacks. In particular, embedded login links have trained users to enter their Facebook and Twitter passwords into other applications.

Android- and Apple-sponsored stores are also quite popular as link targets. The 50 Android applications linked to the Android Market 7 times, and the 50 iOS applications linked to the three Apple stores (iTunes, the App Store, and the Game Center) a collective 23 times. 21 of the web sites targeted at iOS users link to the iTunes and App Store.

# 6. ATTACK PREVENTION

The phishing attacks in this paper all exploit the mobile platform's pervasive lack of application identity indicators. A user cannot reliably tell what application is currently running or what web site is currently loaded in the browser. In the absence of identity indicators, applications and web sites can mimic each other with a high degree of accuracy.

One solution is to permanently dedicate some small portion of the screen to application identity. The operating system would provide an always-present identity bar that displays the name of the current foreground application, and the browser could similarly provide a minimalist, always-present address bar that simply displays the domain in a small font. However, there are three significant problems with this solution: mobile screen space is limited, users ignore security indicators [5], and users will still be conditioned to fall for embedded phishing attacks as long as legitimate applications continue to ask for passwords with embedded login forms.

An alternate solution is for the operating system to support a trusted password entry mechanism. SpoofKiller is a proposal for such a trusted login mechanism [7]. When using SpoofKiller, a user presses the "Home" button when she wants to log in to a web site or application. The operating system then presents the user with a standardized login screen that displays security information and any other

relevant security indicators (e.g., SSL status). The primary challenges for this approach are usability and adoption; users must be convinced to always press the button before supplying a password, and applications and web sites must support this form of password entry.

In general, phishing is an open problem, and the constraints of mobile devices make it more difficult. As an immediate measure, we recommend that companies stop using embedded login forms. Companies like Facebook and Twitter should encourage developers to implement social sharing using their mobile applications instead of embedded logins. Embedded logins exacerbate the problem of training users to ignore security indicators.

## 7. RELATED WORK

Niu et al. identify several aspects of the iPhone's browser user interface that enable phishing attacks [10]. They perform a user study and find that users do not notice missing or spoofed browser user interface elements. Their work applied these techniques to the web-to-web phishing scenario. We build an improved version of their address bar spoofing attack and apply it to new scenarios, such as enabling a web application to spoof a mobile application.

Rydstedt et al. present framing attacks on web sites displayed on mobile phones [13]. They spoof the iPhone's browser address bar to perform their "tapjacking" attacks. Their attacks are all intended for the web-to-web attack scenario, although we believe that their attacks could be extended to mobile applications with WebViews.

Dhamija et al. challenged desktop web browser users to identify phishing attacks, and the best phishing web sites in their data set fooled 90% of the study participants [5]. Simple spoofing attacks (e.g., copying images of SSL indicators) fooled users, and a quarter of their participants only looked at the contents of web sites (e.g., logos and layout) to determine legitimacy. Given their results, we feel it is unlikely that smartphone users could differentiate between our spoofed applications and the legitimate ones.

## 8. CONCLUSION

We examine the threat of phishing on mobile devices. A successful phishing attack has two parts: the user must be conditioned to enter her credentials in a certain setting, and the attacker must be able to imitate that setting. We study real mobile applications and web sites to understand the scenarios in which users enter passwords on mobile phones, and then we propose attacks that subvert these scenarios.

Our analysis of 100 mobile applications and 85 web sites finds that mobile applications and web sites commonly interact in ways that can be spoofed by attackers. Many applications and web sites link to each other for the purpose of social sharing and payment, both of which require the user to enter her authentication credentials in contexts where the user has no way to identify who is receiving those credentials. Users are therefore likely accustomed to switching from one application to another and then entering their passwords into the second application, without any way to verify the authenticity of the second application.

We present 15 phishing attacks that mimic this pattern of interaction. A malicious application can link the user to a social networking or payment web site, and then present the user with a fake login screen. Alternately, an attacker can intercept the interaction and substitute a fake login screen for the intended target. We evaluate how accurately the 15 attacks can mimic legitimate application behavior and conclude that 13 could occur without user detection.

Our analysis suggests that mobile users' passwords for several major sites (notably including Facebook and Twitter) may be at risk. We hope that this research will motivate further research into defenses against mobile phishing.

## 9. REFERENCES


[1] Ettercap. `http://ettercap.sourceforge.net`.

[2] Android bug report. Internet access without permission. `http://code.google.com/p/android/issues/detail?id=8007`.

[3] Canalys. Google's Android becomes the world's leading smart phone platform. Canalys research release 2011/013, 2011.

[4] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner. Analyzing Inter-Application Communication in Android. In *MobiSys*, 2011.

[5] R. Dhamija, J. D. Tygar, and M. Hearst. Why Phishing Works. In *CHI*, 2006.

[6] A. P. Felt, K. Greenwood, and D. Wagner. The Effectiveness of Install-Time Permission Systems for Third-Party Applications. In *WebApps*, 2011, to appear.

[7] M. Jakobsson and W. Leddy. SpoofKiller. `http://www.spoofkiller.com`.

[8] M. Jakobsson, E. Shi, P. Golle, and R. Chow. Implicit Authentication for Mobile Devices. In *HotSec*, 2009.

[9] C. Karlof, J. D. Tygar, and D. Wagner. Conditioned-safe Ceremonies and a User Study of an Application to Web Authentication. In *NDSS*, 2009.

[10] Y. Niu, F. Hsu, and H. Chen. iPhish: Phishing Vulnerabilities on Consumer Electronics. In *UPSEC*, 2008.

[11] PayPal X Developer Network. The Application Showcase Directory. `https://www.x.com/community/ppx/showcase/ap_directory`.

[12] M. Prandini, M. Ramilli, W. Cerroni, and F. Callegati. Splitting the HTTPS Stream to Attack Secure Web Connections. In *IEEE Security & Privacy*, volume 8, Issue 6, December 2010.

[13] G. Rydstedt, B. Gourdin, E. Bursztein, and D. Boneh. Framing attacks on smart phones and dumb routers: tap-jacking and geo-localization attacks. In *WOOT*, 2010.

[14] M. Schultz. Handy Light: Tethering App Camouflaged as Flashlight. `http://appshopper.com/blog/2010/07/20/handy-light-tethering-app-camouflaged-as-flashlight/`.

[15] M. Zalewski. Browser Security Handbook, part 2: Same-origin policy for DOM access. `http://code.google.com/p/browsersec/wiki/Part2#Same-origin_policy`, 2009.