AdSplit
Separating Smartphone Advertising from Applications

SHASHI SHEKHAR, MICHAEL DIETZ, AND DAN S. WALLACH

Shashi Shekhar graduated with an MS in computer science from Rice University and is a software engineer at Google.
shashi.iitg@gmail.com

Michael Dietz is a PhD student at Rice University.
mdietz@gmail.com

Dan S. Wallach is a Professor of Computer Science at Rice University.
dwallach@cs.rice.edu

A wide variety of smartphone applications today rely on third-party advertising services, which provide libraries that are linked into the hosting application. Advertising libraries often need additional permissions, requiring applications to issue requests for additional permissions to their users at install time. This article describes our AdSplit model, where we extended Android to allow an application and its advertising to run as separate processes, under separate user IDs, eliminating the need for applications to request permissions on behalf of their advertising libraries.

Introduction

The smartphone and tablet markets are growing in leaps and bounds, helped in no small part by the availability of specialized third-party applications ("apps"). Whether on the iPhone or Android platforms, apps often come in two flavors: a free version, with embedded advertising, and a pay version without. Both models have been successful in the marketplace. To pick one example, the popular Angry Birds game at one point brought in roughly equal revenue from paid downloads on Apple iOS devices and from advertising-supported free downloads on Android devices [1]. They now offer advertising-supported free downloads on both platforms.

We cannot predict whether free or paid apps will dominate in the years to come, but advertising-supported apps will certainly remain prominent. Already, a cottage industry of companies offer advertising services for smartphone app developers.

Today, these services are simply pre-compiled code libraries, linked and shipped together with an app. This means that a remote advertising server has no way to validate a request it receives from a user legitimately clicking on an advertisement. A malicious app could easily forge these messages, generating revenue for its developer while hiding the advertisements in their entirety. To create a clear trust boundary, advertisers would benefit from running ads separately from their host apps.

In Android, apps must request permission at install time for any sensitive privileges they want to exercise. Such privileges include access to the Internet, access to coarse or fine location information, or even access to see what other apps are installed on the phone. Advertisers want this information in order to better profile users and thus target ads at them; in return, advertisers may pay more money to their hosting apps' developers. Consequently, many apps that require no particular permissions, by themselves, suffer permission bloat, being forced to request the privileges required by their advertising libraries in addition to any of their own needed privileges. Because users might be scared away by detailed permission requests, app developers would also benefit if ads could be hosted in separate apps, which might then make their own privilege requests or be given a suitable one-size-fits-all policy.

Finally, separating apps from their advertisements creates better fault isolation. If the ad system fails or runs slowly, the host apps should be able to carry on without inconveniencing the user. Addressing these needs requires developing a suitable software architecture, with OS assistance to make it robust.

This article primarily focuses on the current state of practice in the Android marketplace, giving a flavor of how we engineered AdSplit as a proof of concept for a better system design.

App Analysis

The need to monetize freely distributed smartphone applications has given rise to many different ad provider networks and libraries. The companies competing for business in the mobile ad world range from established Web ad providers, such as Google's AdMob, to a variety of dedicated smartphone advertising firms.

With so many options for serving mobile ads, many app developers choose to include multiple ad libraries. Additionally, there is a new trend of advertisement aggregators that have the aggregator choose which ad library to use in order to maximize profits for the developer.

Although we're not particularly interested in advertising market share, we want to understand how these ad libraries behave. What permissions do they require? And how many apps would be operating with fewer permissions, if only their advertisement systems didn't require them? To address these questions, we downloaded approximately 10,000 free apps from the Android Market and the Amazon App Store and analyzed them.

Permissions Required

Every ad library requires Internet access, presumably to download the ad content to be displayed. Many libraries want additional privileges to assist in customizing ads. This ranges from location information to the ability to see what else is running on your phone. Presumably, better targeted ads will bring greater revenue to the application developer.

Permission Bloat

In Android, an application requests a set of permissions at the time it's installed. Those permissions must suffice for all of the app's needs and for the needs of its advertising library. We decided to measure how many of the permissions requested are used exclusively by the advertising library (i.e., if the advertising library were removed, the permission would be unnecessary).

Our results, shown in Figure 1, are quite striking: 15% of apps requesting Internet permissions are doing so for the sole benefit of their advertising libraries; 26% of apps requesting coarse location permissions are doing it for the sole benefit of their advertising libraries; and 47% of apps requesting permission to get a list of the tasks running on the phone (the ad libraries use this to check whether the applica-

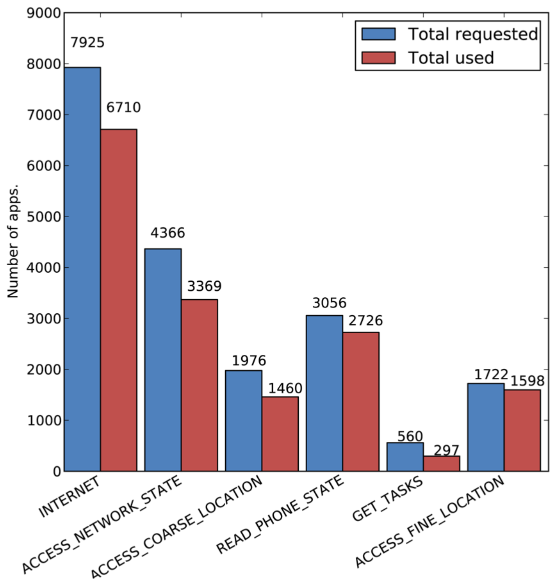


Figure 1: Distribution of types of permissions reduced when advertisements are separated from applications

tion hosting the advertisement is in foreground) are doing so for the sole benefit of their advertising libraries. These results suggest that any architecture that separates advertisements from applications will be able to reduce permission bloat significantly. (In concurrent work to our own, Grace et al. [5] performed a static analysis of 100,000 Android apps and found advertisement libraries uploading sensitive information to remote ad servers. They also found that some advertisement libraries were fetching and dynamically executing code from remote ad servers.)

Design Objectives

The first and most prominent design decision of AdSplit is to separate a host application from its advertisements. This separation has a number of ramifications:

- **Specification for advertisements.** Currently, the ad libraries are compiled and linked with their corresponding host application. If advertisements are separate, then the host activities must contain the description of which advertisements to use. We introduced a method by which the host activity can specify the particular ad libraries to be used.
- **Permission separation.** AdSplit allows advertisements and host applications to have distinct and independent permission sets.
- **Process separation.** AdSplit advertisements run in separate processes, isolated from the host application.
- **Life-cycle management.** Advertisements only need to run when the host application is running, otherwise they can be safely killed; similarly, once the host application starts running, the associated advertisement process must also start running. Our system manages the life cycle of advertisements.
- **Screen sharing.** Advertisements are displayed inside a host app, so if advertisements are separated, there should be a way to share screen real estate. AdSplit includes a mechanism for sharing screen real estate.
- **Authenticated user input.** Advertisements generate revenue for their host applications; this revenue is typically dependent on the amount of user interaction

with the advertisement. The host application can try to forge user input and generate fraudulent revenue, hence the advertisements should have a way to determine whether input events received from the host application are genuine. AdSplit includes a method by which advertising applications can validate user input, validate that they are being displayed on-screen, and pass that verification, in an unforgeable fashion, to their remote server.

The AdSplit Design

Because we want to factor out the advertising code into a separate process/activity, this will require a variety of changes to ensure that the user experience is unchanged.

An app using AdSplit will require the collaboration of three major components: the host activity, the advertisement activity, and the advertisement service. The host activity is the app that the user wants to run, whether a game, a utility, or whatever else. It then "hosts" the advertisement activity, which displays the advertisement. There is a one-to-one mapping between host activity and advertisement activity instances. The UNIX processes behind these activities have distinct user IDs and distinct permissions granted to them. To coordinate these two activities, we have a central advertisement service. The ad service is responsible for delivering UI events to the ad activity. It also verifies that the ad activity is being properly displayed and that the UI clicks aren't forged.

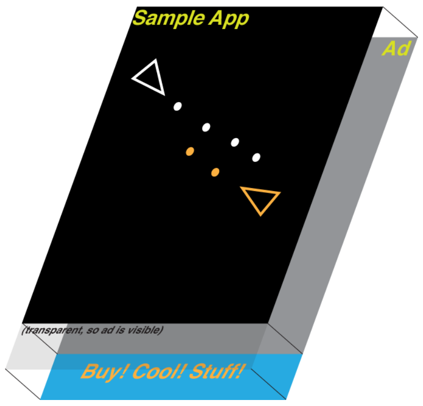


Figure 2: Screen sharing between host and advertisement apps

AdSplit builds on Quire [2], which prototyped a feature shown in Figure 2, allowing the host and advertisement activities to share the screen together. This Quire feature, when combined with a standard Android feature that allows the advertisement activity to detect when its UI is occluded, provides the underpinnings of AdSplit 's UI compositing system.

Permission Separation

With Android's install-time permission system, an application requests every permission it needs at the time of its installation. As we described above, advertising libraries cause significant bloat in the permission requests made by their hosting applications. Our AdSplit architecture allows the advertisements to run as separate Android users with their own isolated permissions. Host applications no longer need to request permissions on behalf of their advertisement libraries.

We note that AdSplit makes no attempt to block a host application from explicitly delegating permissions to its advertisements. For example, the host application might obtain fine-grained location permissions (i.e., GPS coordinates with meter-level accuracy) and pass these coordinates to an advertising library that lacks any location permissions. Plenty of other Android extensions, including TaintDroid [3] and Paranoid Android [8], offer information-flow mechanisms that might be able to forbid this sort of thing if it was considered undesirable. We believe these techniques are complementary to our own, but we note that if we cannot create a hospitable environment for advertisers, they will have no incentive to run in an environment like AdSplit.

Separation for Legacy Apps

A significant number of current apps with embedded advertising libraries would immediately benefit from AdSplit, reducing the permission bloat necessary to host embedded ads. This section describes a proof-of-concept implementation that can automatically rewrite an Android app to use AdSplit. Something like this could be deployed in an app store or even directly on the smartphone itself.

We first built a rewriting system that decompiled an Android app, replacing the internal advertising library with a stub that called out to our AdSplit advertising service. Although we got this working for one specific library, there are a number of problems that would stand in the way of this as a general-purpose solution for AdSplit:

Ad Installation

When advertisements exist as distinct apps in the Android ecosystem, they will need to be installed somehow. We're hesitant to give the host app the necessary privileges to install third-party advertising code. Perhaps an app could declare that it had a dependency on a third-party app, and the main installer could hide this complexity from the user, in much the same way that common Linux package installers will follow dependencies as part of the installation process for any given target.

Ad Permissions

Even if we can get the ad libraries installed, we have the challenge of understanding what permissions to grant them. Particularly when many advertising libraries know how to make optional use of a permission, such as measuring the smartphone's location if it's allowed, how should we decide if the advertisement app has those permissions? Unfortunately, there is no good solution here, particularly not without generating complex user interfaces to manage these security policies.

Ad Unloading

Like any Android app, an advertisement app must be prepared to be killed at any time—a consequence of Android's resource management system. This could have some destabilizing consequences if the hosting app is trying to communicate with its advertisement and the ad is killed. Also, what happens if a user wants to uninstall an advertising app? Should that be forbidden unless every host app which uses it is also uninstalled?

For further details about the implementation of AdSplit 's legacy app support and automatic rewriting, please see our full paper [9].

Alternative Design: HTML Ads

While struggling with the shortcomings outlined above, we hit upon an alternative approach that uses the same AdSplit architecture. The solution is to expand on something that advertising libraries are already doing: embedded Web views.

Ad creators purchasing advertising on smartphones will want to specify their advertisements the same way they do for the Web: as plain text, images, or perhaps as a "rich" ad using JavaScript. Needless to say, a wide variety of tools are available to produce such ads, and mobile advertising providers want to make it easy for ads to appear on any platform (iPhone, Android, etc.) without requiring heroic effort from the ad creators.

Consequently, all of the advertising libraries we examined simply include a WebView within themselves. Most of the native Android advertising code is really nothing more than a wrapper around a WebView. Based on this insight, we suggest that it will be easiest to deploy AdSplit by providing a single advertising app, built into the Android core distribution, that satisfies the typical needs of Android advertising vendors.

Installation becomes a non-issue, since the only advertiser-provided content in the system is HTML, JavaScript, and/or images. We still use the rest of the AdSplit architecture, running the WebView with a separate user ID, in a separate process and activity, ensuring that a malicious app cannot tamper with the advertisements it hosts.

Security permissions are more straightforward. The same-origin policy, standard across the entire Web, applies perfectly to HTML AdSplit. Since the Android WebView is built on the same Webkit browser as the real Web browser app, it has the same security machinery to enforce the same-origin policy.

Keeping all this in mind, we built a new form of WebView specifically targeted for HTML ads: the AdWebView. The AdWebView is a way to host HTML ads in a constrained manner. We introduced two advertisement-specific permissions that can be controlled by the user. These permissions control whether ads can make Internet connections or use geolocation features of HTML5.

When an ad inside an AdWebView requests to load a URL or performs a call to the HTML5 geolocation API, the AdWebView performs a permission check to verify whether the associated advertisement origin has the needed advertisement permission. These advertisement permissions can be managed by the user in exactly the same way they are for any other Web pages.

About the only open policy question is whether we should allow AdSplit HTML advertisements to maintain long-term tracking cookies or whether we should disable any persistent state. Certainly, persistent cookies are a standard practice for Web advertising, so they seem like a reasonable feature to support here as well. AdWebView, by default, doesn't support persistent cookies, but it would be trivial to add.

Policy

Although AdSplit allows for and incentivizes applications to run separately from their advertisements, there are a variety of policy and user experience issues that we must still address.

Advertisement Blocking

Once advertisements run as distinct processes, some fraction of Android users will see this as an opportunity to block advertisements for good. Certainly, with Web browsers, AdBlock and AdBlock Plus are incredibly popular. The Chrome Web store lists these two extensions in its top six with "over a million" installs each. (Google doesn't disclose exact numbers.)

The Firefox add-ons page offers more details, claiming that AdBlock Plus is far and away the most popular Firefox extension, having been installed just over 14 million times, versus 7 million for the next most popular extension. The Mozilla Foundation estimates that 85% of their users have installed an extension (http://blog.mozilla.com/addons/2011/06/21/firefox-4-add-on-users/). Many will install an ad blocker.

To pick one example, Ars Technica, a Web site popular with tech-savvy users, estimated that about 40% of its users ran ad blockers [7]. At one point, it added code to display blank pages to these users in an attempt to cajole them into either paying for ad-free "premium" service, or at least configuring their ad blocker to "white list" the Ars Technica Web site.

Strategies such as this are perilous. Some users, faced with a broken Web site, will simply stop visiting it rather than trying to sort out why it's broken. Of course, many Web sites instead employ a variety of technical tricks to get around ad blockers, ensuring their ads will still be displayed.

Given what's happening on the Web, it's reasonable to expect a similar fraction of smartphone users might want an ad blocker if it was available, with the concomitant arms race in ad block versus ad display technologies.

So long as users have not "rooted" their phones, a variety of core Android services can be relied upon by host applications to ensure that the ads they're trying to host are being properly displayed with the appropriate advertisement content. Similarly, advertising applications (or HTML ads) can make SSL connections to their remote servers, and even embed the remote server's public key certificate, to ensure they are downloading data from the proper source, rather than empty images from a transparent proxy.

Once a user has rooted their phone, of course, all bets are off. While it's hard to measure the total number of rooted Android phones, the CyanogenMod Android distribution, which requires a rooted phone for installation, is installed on roughly 722,000 phones—a tiny fraction of the hundreds of millions of Android phones reported to be in circulation. Given the relatively small market share where such hacks might be possible, advertisers might be willing to cede this fraction of the market rather than do battle against it.

Consequently, for the bulk of the smartphone marketplace, advertising apps on Android phones offer greater potential for blocking-detection and blocking-resistance than advertising on the Web, regardless of whether they are served by in-process libraries or by AdSplit. Given all the other benefits of AdSplit, we believe advertisers and application vendors would prefer AdSplit over the status quo.

Permissions and Privacy

Leaving aside whether it's legal for advertisers to collect sensitive information such as a user's precise location, we could always invent technical means to block this as a matter of policy. Unfortunately, a host app could always make its own requests, under its own authority, that violate the user's privacy and pass these into the AdSplit advertising app. Can we disincentivize such behavior? We hope that, if we can successfully reduce apps' default requests for privileges that they don't really need, then users will be less accustomed to seeing such permission requests. When they do occur, users will push back, refusing to install the app. (Reading through the user-authored comments in the Android Market, many apps with seemingly excessive permission requirements will have scathing comments, along with technical justifications posted by the app authors to explain why each permission is necessary.)

Furthermore, if advertisers ultimately prefer the AdSplit architecture, perhaps due to its improved resistance to click fraud and so forth, then they will be forced to make the tradeoff between whether they prefer improved integrity of their advertising platform, or whether they instead want less integrity but more privacy-violating user details.

Conclusion

AdSplit touches on a trend that will become increasingly prevalent over the next several years: the merger of the HTML security model and the smartphone application security model. Today, HTML is rapidly evolving from its one-size-fits-all security origins to allow additional permissions, such as access to location information, for specific pages that are granted those permissions by the user. HTML extensions are similarly granted varying permissions rather than having all-or-nothing access [6].

On the flip side, iOS apps originally ran with full, unrestricted access to the platform, subject only to vague policies enforced by human auditors. Only access to location information was restricted. In contrast, the Android security model restricts the permissions of apps, with many popular apps running without any optional permissions at all. Despite this, Android malware is a growing problem, particularly from third-party app stores (see, e.g., [4, 10]). Clearly, there's a need for more restrictive Android security, more like the one-size-fits-all Web security model.

While the details of how exactly Web apps and smartphone apps will eventually combine, our findings show where this merger is already underway: when Web content is embedded in a smartphone app. Well beyond advertising, a variety of smartphone apps take the strategy of using native code to set up one or more Web views and then do the rest in HTML and JavaScript. This has several advantages: it makes it easier to support an app across many different smartphone platforms. It also allows authors to quickly update their apps, without needing to go through a third-party review process.

These trends, plus the increasing functionality in HTML5, suggest that "native" apps may well be entirely supplanted by some sort of "mobile HTML" variant, not unlike HP/Palm's WebOS, where every app is built this way.

Maybe this will result in an industry battle royale, but it will also offer the ability to ask a variety of interesting security questions. For example, consider the proposed "Web intents" standard (http://webintents.org/). How can an "external" Web intent interact safely with the "internal" Android intent system? Both serve essentially the same purpose and use similar mechanisms. We, and others, will pursue these new technologies toward their (hopefully) interesting conclusions.

References

[1] T. Cheshire, "In Depth: How Rovio Made Angry Birds a Winner (and What's Next)," *Wired,* Mar. 2011: http://www.wired.co.uk/magazine/archive/2011/04/features/how-rovio-made-angry-birds-a-winner.

[2] M. Dietz, S. Shekhar, Y. Pisetsky, A. Shu, and D.S. Wallach, "Quire: Lightweight Provenance for Smart Phone Operating Systems," 20th USENIX Security Symposium, San Francisco, CA, Aug. 2011.

[3] W. Enck, P. Gilbert, C. Byung-gon, L.P. Cox, J. Jung, P. McDaniel, and A.N. Sheth, "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones," *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation* (OSDI '10), Oct. 2010, pp. 393–408.

[4] A.P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A Survey of Mobile Malware in the Wild," 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM '11), Chicago, IL, Oct. 2011.

[5] M. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi, "Unsafe Exposure Analysis of Mobile In-App Advertisements," 5th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '12), Tucson, AZ, Apr. 2012.

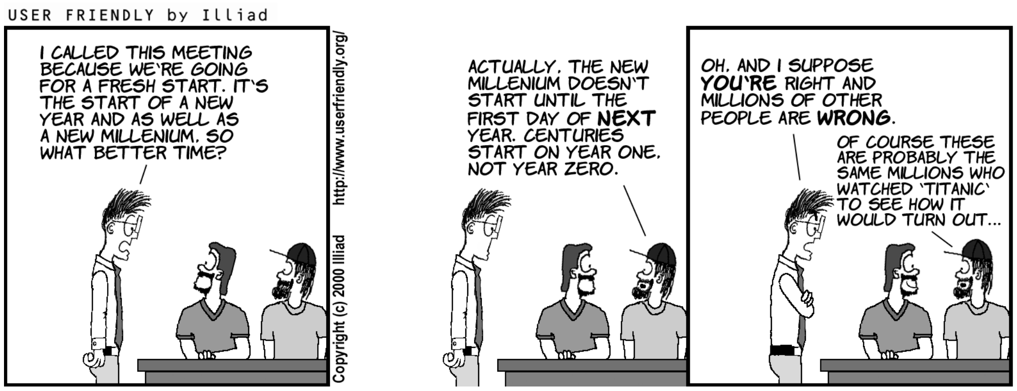
[6] L. Liu, X. Zhang, G. Yan, and S. Chen, "Chrome Extensions: Threat Analysis and Countermeasures," 19th Network and Distributed System Security Symposium (NDSS '12), San Diego, CA, Feb. 2012.

[7] L. McGann, "How Ars Technica's 'Experiment' With Ad-Blocking Readers Built on Its Community's Affection for the Site," Nieman Journalism Lab, Mar. 2010: http://www.niemanlab.org/2010/03/how-ars-technica-made-the-ask-of-ad-blocking-readers/.

[8] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos, "Paranoid Android: Zero-Day Protection for Smartphones Using the Cloud," Annual Computer Security Applications Conference (ACSAC '10), Austin, TX, Dec. 2010.

[9] S. Shekhar, M. Dietz, and D. Wallach, "Adsplit: Separating Smartphone Advertising from Applications," 21st USENIX Security Symposium, Bellevue, WA, Aug. 2012.

[10] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets," 19th Network and Distributed System Security Symposium (NDSS '12), San Diego, CA, Feb. 2012.


USER FRIENDLY by Illiad
I CALLED THIS MEETING BECAUSE WE'RE GOING FOR A FRESH START. IT'S THE START OF A NEW YEAR AND AS WELL AS A NEW MILLENIUM. SO WHAT BETTER TIME?
http://www.userfriendly.org/
Copyright (c) 2000 Illiad
ACTUALLY, THE NEW MILLENIUM DOESN'T START UNTIL THE FIRST DAY OF NEXT YEAR. CENTURIES START ON YEAR ONE, NOT YEAR ZERO.
OH, AND I SUPPOSE YOU'RE RIGHT AND MILLIONS OF OTHER PEOPLE ARE WRONG.
OF COURSE THESE ARE PROBABLY THE SAME MILLIONS WHO WATCHED 'TITANIC' TO SEE HOW IT WOULD TURN OUT...