Can Mobile learn from the Web?

Kapil Singh
IBM T.J. Watson Research Center
kapil@us.ibm.com

***Abstract*—The tremendous growth in popularity of smartphones has been closely matched by increased efforts to harness their potential. This has lead to the development of powerful mobile operating systems that provide novel programming platforms for the creation of rich mobile applications. To support these new paradigms, developers are now asked to spend considerable effort in replicating functionality, usually already available in web applications, for the native applications.**

In this paper, we observe that web applications closely simulate the design of native applications and the web origin can act as a more reliable authentication identifier for third-party application content. We consequently argue in favor of using web applications as the default for mobile platforms and propose a practical approach to extend web applications with native application-like capabilities. Central to our approach is a browser-based permission model that effectively manages permissions both at install time and at runtime, and supports dynamic user policies. We discuss security and non-security challenges of realizing this approach.

I. Introduction

The rapid growth of mobile computing and increasing adoption of smartphones has resulted in the evolution of rich mobile platforms, such as Android and iOS. While these platforms derive much of the functionality from traditional desktop operating systems, they also introduce security mechanisms to satisfy the new requirements imposed by the mobile environment. One such novel mechanism is the application-centric permission model developed for Android that controls access to sensitive data (such as contact list) and resources (such as camera) available on the device.

While the permission model provides users with an ability to control access to their data and resources, it also introduces a new programming paradigm where applications must follow a defined structure and are platform-specific. For instance in Android the applications are typically written in Java and use a component-based architecture to modularize their functionality. This introduces an additional burden on application developers who need to learn and understand a new programming paradigm.

With rich features, such as camera and GPS, readily available on mobile devices, there is an increasing motivation for individuals and businesses to harness these resources to provide enhanced functionality in their applications. Over the years, a large majority of these individuals and businesses have provided their online services on the Web. However, instead of just enhancing their current applications to use the mobile features, they are asked to redevelop their applications in the new platform-specific programming paradigms. For example, a bank has to individually develop its application for Android and iOS, when it already has a potentially well-used, and hence well tested, web application.

In this position paper, we show that there is a close correspondence between the new mobile paradigms and the existing web platform in the context of application development. We use the Android platform as our representative example in the paper; the basic concepts apply to other mobile platforms as well. Consequently, we argue that the web platform can be adapted to satisfy the requirements of the mobile platform, effectively resulting in a natural adaptation of web applications to have similar capabilities as the native mobile applications.

We argue that the web origin act as a natural identification token for applications instead of a signature-based approach used in native applications. While new programming platforms, such as PhoneGap [1], enable the use of web technologies (HTML, JavaScript and CSS) in native applications, all files are stored locally on the device with the loss of origin information. A local origin is associated with all the files irrespective of their source. As a result, it suffers from the same developer-authentication issue as the signature-based approach used by Android (Section II).

Based on our learnings, we propose that web applications running in a browser be given access to the the data and features exposed by the underlying platform APIs. To monitor access, we propose a browser-based permission model similar in concept to the Android model. From the perspective of permission enforcement, a browser-based approach is inherently advantageous over the current platform-based approach as it enables a dynamic permission model where the users can modify permissions at runtime (Section III).

To realize the concepts proposed in the paper, we also discuss a candidate design of a prototype system that can be readily deployed without any major modifications to the current web setup. Our proposed design uses browser extensions that proxy all API calls to the underlying operating system (Section III).

This proposal makes the following contributions:

- It provides a correlation between the mobile world and the web world from the perspective of application development.
- It proposes that web origin should act as the identification token for browser-based mobile applications.
- It proposes a browser-based permission model and provides a practical approach for its implementation.

II. Mobile Apps – a shadow of the Web Apps?

In this section, we draw parallels between the security context of native mobile applications and web applications. We consider three major building blocks as our comparison vectors: application isolation, application origin and inter-application communication.

A. Isolation

In Android, each application runs within its own security environment contained within the application sandbox. Android is built on top of the Linux kernel and takes advantage of the Linux user-based protection for sandboxing the applications. It assigns a unique user ID (UID) to each Android application and runs it as that user in a separate process. Applications also get a dedicated part of the filesystem in which they can write private data, including databases and raw files.

Browsers have also recently transitioned into using a multi-process design that runs applications in different processes to keep them isolated from each other and sandboxed for restricting access to the underlying operating system [2] [3] [4]. Mobile browsers have started to catch up on this trend [5]. This multi-process design helps the browser to isolate crashes and vulnerabilities to specific applications instead of the complete browser. Irrespective of its design, the browser is responsible for isolating the web applications that is governed by the same origin policy (SOP) [6].

B. Origin for applications

The concept of *origin* intuitively represents an authentication identity of a particular application. A compromise of this identity can potentially lead to data leakage or system compromise by malicious applications that can masquerade a trusted application.

In the Android system, each application is signed by the developer of the application and this signature is used by the system to assign unique UIDs for isolation[1]. Effectively, the developer's signature uniquely identifies the origin of the application.

Android allows self-signed developer signatures without any need for a certification authority. While it is convenient and economical for the application developers, it does not provide reliable authentication of a developer's identity. As a result, the end user is ill-equipped to make an install-time decision of adding an application without a verified developer.

On the other hand, web applications run within the context of the web browser and have no direct access to the resources available on the device. Resource access across applications is determined by the SOP, that defines the web origin as the triple of `<protocol, domain, port>` [6].

Considering their simpler representation and users' day-to-day familiarity of web URLs in comparison to signatures, we believe that the web origins is more intuitive representation from the users' perspective. Moreover, the web infrastructure ensures that the origin is rightly associated with the content of the web application[2]. The use of HTTPS further ensures server authenticity and content integrity. The users can further leverage free URL reputation services [7] [8] to filter out malicious application domains. Therefore, we propose to use the web origin as the authentication identifier for our proposed browser-based hybrid applications discussed in Section III.

C. Inter-application communication

Android platform provides a collaborative application environment where an application can leverage existing data and services provided by other applications. This promotes development of rich applications that enables functionality reuse with reduced developer effort. Android supports this by means of a message passing system that enables communication within and across application boundaries. A central component of this system are the *Intent* objects that are typed interprocess messages used to link applications. Intents are directed to particular applications or system services, or broadcast to applications that are subscribed to a particular intent type.

On the web front, postMessage [9] provides a secure communication medium between web pages of different origin. There is a close correlation between Intents and postMessage as both allow only serializable objects to be passed. However, there are also some subtle differences. While postMessage is directed to a specific target object using the object's handle, Intents provide a much richer medium to specify its target by additionally supporting *implicit* communication where the target is determined by the Android platform based on the operation that needs to be performed.

In addition to their logical similarities, they have also been shown to suffer from similar issues due to insecure developer practices. While postMessage requires explicit specification of target by the sender and verification of sender by the receiver to prevent potential attacks [10], Intents have been shown to be vulnerable to similar spoofing and data leakage issues [11].

Recently the concept of Web Intents has been proposed as a framework for inter-application communication [12]. While it has currently not been implemented by the browsers and is available as a JavaScript shim, it further bridges the gap between the mobile and web platforms. However, Web Intents uses HTML5 features and hence is not backward compatible with older browsers.

III. Browser-based permission model

The previous section drew parallels between the mobile and the web platforms from the perspective of their applications. In this section, we leverage the similarities to propose a hybrid application paradigm that uses web applications *running in the browser* with enhanced capabilities similar to native applications. This is in contrast to the "hybrid" applications that are developed using web technologies, but are hosted by the operating system as native applications [1]. Note that we would use the term hybrid to represent our proposed

[1]Applications signed with same developer key can optionally declare a shared UID in their manifest to access each other's resources using a shared sandbox

[2]Attacks, such as DNS rebinding and man-in-the-middle attacks, can still modify the web content, but are much harder to execute.

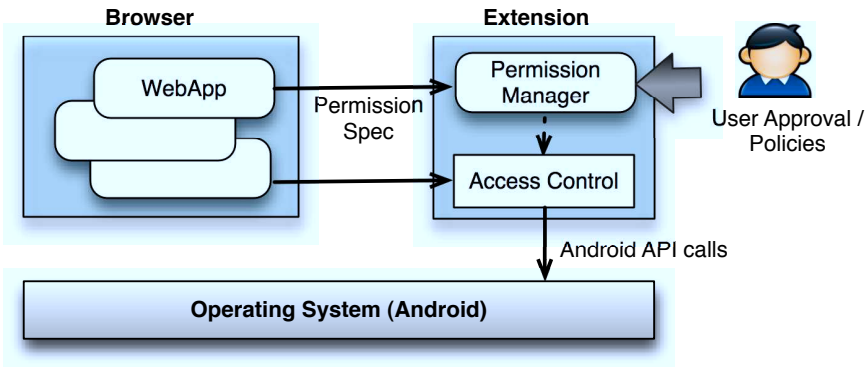

Fig. 1. High-level architecture of our proposed design.

application design further in the text. By hosting the hybrid applications within the web browser, we can leverage the web origin as the authentication identifier for the applications. Our position is to use this hybrid application design as the default standard for mobile platforms.

We propose that web applications are assigned privileged capabilities to access device data and resources. These capabilities would allow the applications to access advanced hardware and software features, as well as local and served data, exposed by the mobile platform. Taking mobile banking as an example, the banking web application would be able to access to the camera on the device for uploading the picture of a deposit check to the banking site. This would enable billions of existing web applications to readily include use of the mobile features, with minimal modifications (potentially only a few lines of API calls). In contrast, native applications that provide the same functionality as the web applications would need to be developed independently with explicit development effort.

A desirable side-effect of using web applications is reduced management and resource usage as compared to the installed native applications. Web applications are accessed on need-to-use basis in comparison to the persistent native applications. As the number of installed applications continue to rise, keeping track of all installed applications can become cumbersome and a major inconvenience for the users.

We believe that similar to the current mobile platforms, access to the device's resources and data should still be controlled using a application-specific permission model. However, the enforcement of such a model should be done by the browser as the semantics and context of a hybrid application are well exposed within the browser. A browser-based permission enforcement would allow dynamic granting and revocation of permissions at application runtime. This is an advantage over traditional native application where permissions cannot be altered after application installation and can only be revoked by uninstalling the application.

In essence, any potential design of browser-centric hybrid applications would require the browser to first expose the underlying Android APIs to the applications and then grant access to these privileged APIs based on the permissions granted to the requesting application.

A. Proposed system design

Web browsers, by default, do not provide web applications with any access to the underlying operating system's resources such as the file system. However, browser extensions are given direct access to such resources. Our proposed implementation would use extensions as a medium to invoke the platform APIs for resource access[3]. An alternate design would be to include this functionality into the browser core, however, it would involve modifications to the browser code thereby limiting its near-term large-scale deployment.

Figure 1 shows a high-level architecture of our proposed design. The browser extension effectively acts as a proxy to provide controlled access to the platform APIs. Similar to Android's native applications, the access given to a hybrid application is governed by a set of permissions that are approved for that particular application.

Realization of the permission model: In our design, permissions are managed by the *Permission Manager* module that is common for all applications. Instead of using a manifest file to specify the request permissions (as done for native applications in Android), hybrid applications include their permission specification as content in the HTTP cookies or as a part of the page's content marked by a special tag. The application can also optionally include an expiration time for the permissions (for a cookie-defined specification, it can effectively represent expiration of the cookie). The browser extension extracts this information and presents it to the user for approval. Once approved, these permissions can be cached by the permission manager for use by subsequent invocations of the hybrid applications.

The use of a single Permission Manager for all applications could potentially facilitate tracking of permission misuse across applications (such as detection of confused deputy attack [14]) as it has a single view of permissions for all communicating applications. However, such detection would also require monitoring of postMessage calls to track the flow of information.

An application invokes the extension and specifies the

[3]While current mobile browsers have only started to support extensions [13], we expect that the support would be added in most browsers in near future.

platform API to be called along with the corresponding parameters. The extension verifies with the Permission Manager if the requesting origin is allowed to make the API call and if allowed, it makes the API call on the application's behalf.

Our design is analogous to Mozilla's WebAPI effort [15] and Google's Chrome Web Store [16], however, it uses a simpler transformation by extracting the underlying platform's permission model and by mapping the requests directly to the platform's APIs. WebAPI is a more generic effort that targets multiple platforms, while Chome Web Store is targeted for the Chrome browser. Both these efforts use a strict developer verification approach (as used by iOS), while we rely on the web origins for server authentication.

Dynamic User Policies: Our system also has a provision for user policies that can be used to blacklist or whitelist certain origin-to-permission mappings. Specification of such policies in advance is made possible due to the fact that users already have trust relationships with certain web domains (for example, a user can trust his bank's site to have access to his camera). The user policies can also be used to dynamically modify permissions at runtime.

IV. Discussion

We envision that browser-based hybrid applications would become the default on mobile platforms. However, there is an constant debate on the use of native applications in comparison to web applications citing performance and usability considerations [17]. At the same time, the popularity of web applications has not diminished even with the presence of native applications [18]. We also expect the performance of web applications to improve with growing efforts [19] [20] to address this issue.

There are privacy risks associated with allowing web applications to have access to a device's private data as the application is hosted remotely on a web server. However, such risks even exist for native applications as most applications are given a permission to access the Internet and allowed to communicate with any external server *without any restrictions*. Traditionally SOP allowed a web application to communicate only with its own server, however, such restrictions have recently been removed allowing cross-domain server requests. Further research is needed to address this privacy issue.

Allowing hybrid applications to dynamic request additional permissions at runtime can potentially lead to user frustration. This can eventual lead to a security trap as users can end up accepting all such requests by default. Specific provisions can be made in the Permission Manager to define policies for ignoring such requests; such policies can be made domain-specific and/or privilege-specific. The usability aspect of defining such policies comprehensively remains an open research problem.

The hybrid applications rely on the security provided by the web browser and any bug or vulnerability in the browser could potentially circumvent the isolation provided by the SOP. Exposing native APIs to the web origins raises the stakes in case of a security breach, however, additional defences developed for the web [4] [21] are also readily applicable to hybrid applications. Native applications have been shown to suffer from similar security issues [14] [22] and it is a matter of debate whether mobile security solutions are more mature than their web counterparts.

One major limitation for server-hosted applications is availability in case of poor or no network connectivity. The application can improve offline support by caching content in the browser. Depending on the application, this might limit still functionality that is provided at the server side and not rendered in the client-side component of the application.

References

[1] "Adobe PhoneGap," http://www.phonegap.com.

[2] "Whats New in Internet Explorer 8," http://msdn.microsoft.com/en-us/library/cc288472.aspx.

[3] C. Reis and S. D. Gribble, "Isolating Web Programs in Modern Browser Architectures," in *Proceedings of the 4^{th} ACM European Conference on Computer systems (EuroSys)*, Nuremberg, Germany, Mar. 2009.

[4] C. Grier, S. Tang, and S. T. King, "Secure Web Browsing with the OP Web Browser," in *Proceedings of the 29^{the} IEEE Symposium on Security and Privacy*, Oakland, CA, May 2008.

[5] L. Parfeni, "Chrome for Android Under the Hood: Multi-Process, V8 but No WebGL or Sandboxing," http://news.softpedia.com/news/Chrome-for-Android-Under-the-Hood-Multi-Process-HTML5-V8-But-No-WebGL-or- Sandboxing-251491.shtml.

[6] J. Ruderman, "Same Origin Policy for JavaScript," http://www.mozilla.org/projects/security/components/same-origin.html.

[7] "Google Safe Browsing API," http://code.google.com/apis/safebrowsing/.

[8] "Microsoft SmartScreen Filter," http://windows.microsoft.com/en-US/internet-explorer/products/ie-9/features/smartscreen-filter.

[9] A. Barth, C. Jackson, and J. C. Mitchell, "Securing Frame Communication in Browsers," in *USENIX Security Symposium*, San Jose, CA, Jul. 2008.

[10] "PostMessage," https://developer.mozilla.org/en/DOM/window.postMessage.

[11] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner, "Analyzing Inter-application Communication in Android," in *Proceedings of the 9^{th} International Conference on Mobile Systems, Applications, and Services (MobiSys)*, Bethesda, MD, Jun. 2011.

[12] "Web Intents," http://webintents.org/.

[13] B. Hitching, "Social Browsing on your iPhone with Safari Browser Extensions," http://hitching.net/2010/06/28/social-browsing-on-your-iphone-with- safari-browser-extensions/.

[14] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android Permissions Demystified," in *Proceedings of the 18^{th} ACM Conference on Computer and Communications Security (CCS)*, Chicago, IL, Nov. 2011.

[15] "WebAPI," https://wiki.mozilla.org/WebAPI.

[16] "Chrome Web Store," http://chrome.google.com/webstore.

[17] T. Bray, ""Web" vs, "Native"," http://www.tbray.org/ongoing/When/201x/2011/06/14/Native-vs-Web.

[18] "Mobile-enabled Website: Mobile Web Apps vs Native Apps," http://k2bindia.com/blog/mobile-enabled-website-mobile-web-apps-vs-native-apps/.

[19] "SPDY: An experimental protocol for a faster web," http://dev.chromium.org/spdy/spdy-whitepaper.

[20] "Let's make the web faster," http://code.google.com/speed/.

[21] M. V. Gundy and H. Chen, "Noncespaces: Using Randomization to Enforce Information Flow Tracking and Thwart Cross-Site Scripting Attacks," in *Proceedings of the 16^{th} Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2009.

[22] M. Grace, Y. Zhou, Z. Wang, and X. Jiang, "Systematic Detection of Capability Leaks in Stock Android Smartphones," in *Proceedings of the 19^{th} Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2012.