Poster: Detecting Unexpected Behaviors in HTML5 Mobile Apps using Difference in Execution Context

Jian Mao*, Yaoqi Jia†, Xinshu Dong‡, Yue Chen*, Ruilong Wang*, and Zhenkai Liang†

*School of Electronic and Information Engineering, Beihang University, China
†School of Computing, National University of Singapore
‡Advanced Digital Sciences Center, Singapore
Email: maojian@buaa.edu.cn

I. Introduction

HTML5, JavaScript, and CSS are the standard development technology used in web applications. In a recent development, they are extended as a popular way to develop cross-platform local applications, especially mobile applications. For example, Apache Cordova provides a middle layer that enables developer to use web development technology to develop mobile applications that run across all mobile platforms, such as Apple's iOS, Android, and Windows Phone.

This new development paradigm (we refer to it as HTML5 mobile app) quickly gains popularity because they provide an efficient way for developing cross-platform applications, which significantly saves engineering efforts of mobile development that target a large range of devices with different mobile operating systems. Moreover, since web technology is used in mobile application development, it allows an enterprise to keep a common code base between its website and its mobile app. This design greatly reduces the efforts of software maintenance.

Though there are significant software engineering benefits, HTML5 mobile apps have their new concerns on integrity and data privacy. Prevalent attack vectors on the web, such as cross-site scripting (XSS), can now be leveraged to compromise HTML5 applications. Once a piece of malicious script enters into a victim HTML5 mobile application, it assumes all the privileges of the victim application. To make things worse, the privilege of an HTML5-based mobile app is more powerful than traditional web applications: the browser engines is extended with more capability for accessing local resources (such as file system, camera, and microphone).

Our Observations. By design, a program behavior, such as accessing device information, can only appear in certain contexts of an HTML5 mobile app. In particular, it should be generated by specific browser behavior sequences and triggered by specific browser UI components. Such behavior sequence and UI components form the execution context, which helps to distinguish malicious behavior from benign ones. For example, an app may only access device UUID during the initialization after installation; Device UUID access made by injected code may happen any time.

II. Approach

We propose a dynamic behavior diagnosis solution for HTML5 mobile apps on Android. We aim to provide an infrastructure that monitors run time behaviors of such applications, encompassing their internal states, their interactions with users and network, as well as their access to user data on the devices. Such an infrastructure will establish a chain of events and form the contexts needed for detecting malicious behaviors. In particular, we refer to the dependency on DOM nodes and triggering events as the *static context*, and the sequence of with app behaviors as the *dynamic context*. To better explain our solution, we elaborate how we maintain the information for static and dynamic contexts below.

- *Static Context.* The static context provides the fundamental correlation for an event or action. For example, suppose a user clicks on a button, and an device UUID access is generated in the web application. In this scenario, the immediately involved DOM node for the button, the click event, and the event handler constitute the static context for the access. It informs of how the access is generated, and which parties are involved. In terms of malicious behavior detection, the static context enables us to distinguish two seemingly equivalent device UUID requests that are generated differently, one by normal user interactions and another, for instance, by injected scripts.

  To monitor the static context for asynchronous events in HTML5 mobile applications, we maintain a virtual stack structure in WebView. A virtual stack works in a similar way as a call stack; it pushes a new entry onto the stack when an event starts being processed, and pops it out when the processing completes. However, it augments the call stack by storing

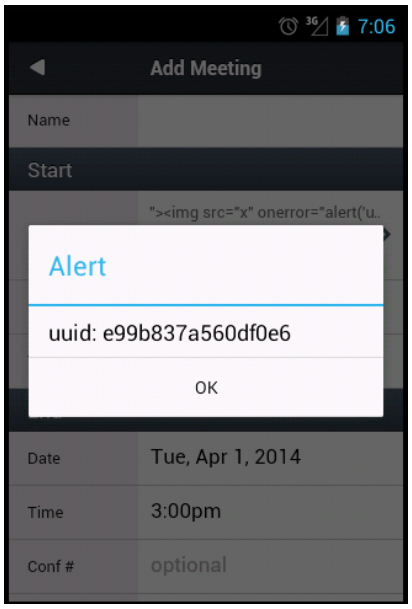

Figure 1: Injected malicious scripts exfiltrate device information from TripCase.

more details of the events, DOM nodes and code involved. In the previous example, when the access request is generated, the virtual stack includes entries corresponding to a user click event, the DOM node for the button, and the JavaScript function serving as the event handler. Such information allows our solution to query the details on the context where an action occurs.

However, this is not sufficient in capturing more stealthy malicious behaviors. As we can see, an important piece of information here is the JavaScript event handler function. This function in fact determines the final access request generated. A malicious script may modify the function to generate a complete different request, such as exfiltrate user data to attackers' servers. Even if this event handler function has been tampered, the static context will still appear normal. Thus, we introduce the dynamic context to resolve such insufficiency.

- *Dynamic Context.* We specifically develop the dynamic context to associate asynchronous events that do not exhibit a direct causal relationship. For example, a JavaScript function may have been tampered with by another piece of script X first, and then only at a later point it is invoked. The dynamic context thus correlates the static context at the time of the former event, script modification, and the static context at the latter event, function invocation. Thus, the dynamic context re-connects the parts missing in the static context, and exposes other parties involved (script X) that contribute to the generation of the HTTP request.

  We establish the correlation between different static contexts with the shared object references. For example, in the aforementioned example, the shared object reference is the JavaScript event handler function. Consider another example where an XMLHttpRequest is sent to a web server asynchronously. When the corresponding response arrives, the dynamic context connects the present one to the original static context where the XMLHttpRequest was generated. Here the shared object reference is the XMLHttpRequest itself. With the dynamic context, we enable a much more comprehensive grasp of correlation between asynchronous events and actions, which enables precise behavior diagnosis as demonstrated in our results.

III. Results

We have made a preliminary implementation based on the CyanogenMod project, as well as a preliminary evaluation of our approach.

We modified the Android platforms browser component, WebView, for monitoring mobile applications' internal events, such as DOM access and JavaScript execution, and its access to user data and device resources, such as geolocation and camera. We correlates the events to form the static and dynamic contexts.

In our evaluation, we use real-world app samples such as HealthTap and TripCase, as well as synthesized examples that are vulnerable to script injection. For example, TripCase is a popular mobile App built with HTML5, which helps users to record their trip details (e.g, flight, hotel, etc.). Its Android version has over 100,000 downloads. The app is migrated from its web application. An XSS vulnerability in the app allows injected JavaScript code to gain access to local resources, such as obtaining device information, reading contact list, or accessing files. Figure 1 shows the screen shot where an injected malicious script obtains device information. In our experiment, our approach successfully detects the unexpected behaviors based on contextual differences.

In summary, we present an approach to detect unexpected behaviors in HTML5 mobile apps on the Android platform. The key contribution is the identification of important contextual information in execution of HTML5 mobile apps, the techniques to extract the contexts, and the application to detect unexpected behaviors. Our preliminary results show the potential of the techniques on real-world problems.

Acknowledgment. This work was supported in part by the Beijing Natural Science Foundation (No. 4132056), the National Key Basic Research Program (NKBRP) (973 Program) (No. 2012CB315905), Natural Science Foundation of China (No. 61272501, 61173154, 61003214).