Poster: AdHoneyDroid – Capture Malicious Android Advertisements

Dongqi Wang[1], Shuaifu Dai[1,2], Yu Ding[1], Chen Li[1], Xinhui Han[1]
[1]Peking University, Beijing, China
[2]CNCERT, Beijing, China
{imdq, dingyu.icst, icst-lichen, hanxinhui}@pku.edu.cn, daishuaifu@chanct.com

ABSTRACT

In this paper we explore the problem of collecting malicious smartphone advertisements. Most smartphone app contains advertisements and also suffers from vulnerable advertisement libraries. Malicious advertisements exploit the ad library vulnerability and attack victim smartphones. Similar to the traditional honeypots, we need an effective way to capture malicious ads. In this paper, we provide our approach named AdHoneyDroid. We build a crawler to gather apps on the android marketplaces and manually collect ad libraries and their vulnerabilities. Then AdHoneyDroid executes the apps and detects malicious advertisements. In our approach, we adopt the idea of API sandbox and TaintDroid to detect the attack event. We store the malicious advertisements in a database for future analysis. Malicious ads can help security analysts have a better understanding of current mobile attacks and also disclose the attack payloads.

Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: Invasive software

Keywords

Malicious Ads; Android; Attack Detection

1. INTRODUCTION

Most smartphone apps contains advertisements [1]. Malicious advertisements are threatening more and more smartphones. Millions of Android devices are suffering from vulnerabilities such as the JS-Binding-Over-HTTP Vulnerability [2, 3]. To understand such vulnerabilities and attack events, we need an effective way to detects malicious advertisement attacks and collect these malicious ads.

Why advertisement library vulnerability has so much impact on mobile security? First, almost each mobile app contains at least one advertisement library and every smartphone runs apps from time to time. The vulnerability of advertisement library can directly put user's privacy in a dangerous place. Second, most advertisement library provides direct Javascript to JAVA interfaces to the developer. Thus the vulnerability can be directly used to invoke sensitive system functions and grant attacker a higher privilege to control the victim smartphone. Third, a large amount of smartphone users seldom update their devices' operating system. Thus the attacker can control the pwned phones for a long period of time. So the vulnerability of ad library is a critical threat to the mobile users.

Similar to the traditional honeypots, we need a mobile honeypot to capture the malicious advertisements. The mobile honeypot should be able to identify and capture malicious advertisements among all incoming advertisements. To achieve this goal, mobile honeypot needs to detect malicious ads attacks. So the main challenge is: how to detect the incoming malicious ads attacks.

To achieve these goals, we design and build a prototype named AdHoneyDroid. AdHoneyDroid detects malicious ads attacks by using TaintDroid [4] and API sandboxing. We manually analyze disclosed ads library vulnerabilities and generate the vulnerability signature on the API function level. For each app, AdHoneyDroid automatically decompile the apk file and generate ADs url whitelist and function call blacklist. AdHoneyDroid identifies incoming attacks by using the manually generated known vulnerability signature, automatically generated app API blacklists, and privacy leakage detection by using TaintDroid.

Assumption and Adversary Model.

In this paper we assume that the ad libraries do not contain Javascript codes and all the javascript codes executed in the ad libraries come from incoming advertisements. The attacker attacks victim smartphone by injecting malicious javascripts into the advertisements to trigger the vulnerability of advertisement libraries.

2. SYSTEM DESIGN

AdHoneyDroid collects Android apps from markets and executes them in a monitored environment and capturing incoming malicious ads. The workflow of AdHoneyDroid is shown in figure 1. There are four key steps in the workflow: crawling, preprocessing, running and logging.

Crawling.

We build a simple crawler to automatically collect android app apk files from Google Play [5] market. Due to the complexity of aggregator, we only consider the apps which only contain one single ad library. So we manually collected

CCS '14 November 3 - 7,2014,Arizona,USA
ACM 978-1-4503-2957-6/14/11.
http://dx.doi.org/10.1145/2660267.2662395 .

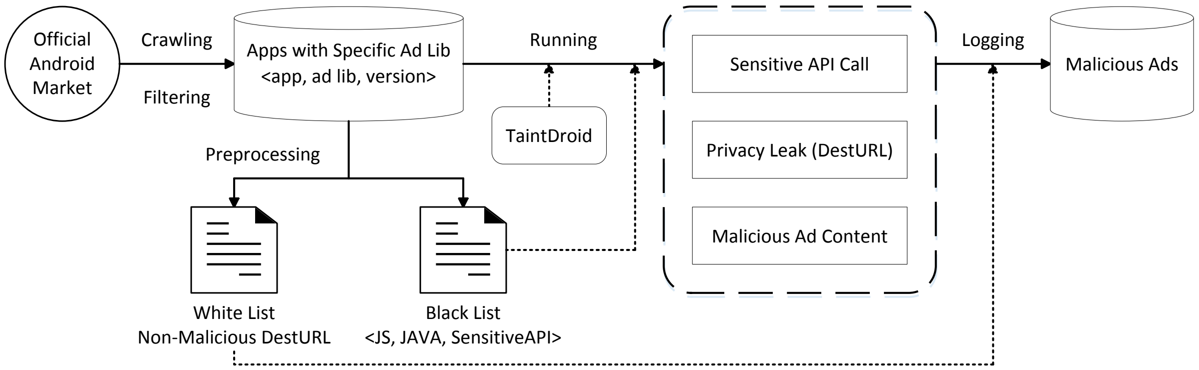

Figure 1: Workflow of AdHoneyDroid

these simple ad libraries. AdHoneyDroid decompresses the apk file and filters out the apps which contain ad aggregator or more than one ad libraries. So after the crawling and filtering step, AdHoneyDroid keeps a set of apps which only contains one specific ad lib.

Preprocessing.

In the preprocessing step, AdHoneyDroid decompiles the collected apps and generate a whitelist and a blacklist. The whitelist contains non-malicious ad URLs and the blacklist contains potential paths from JS interfaces to sensitive API functions.

The whitelist is used to decrease the false positive rate, because every single ad library interacts with certain servers on library initialization. So the whitelist contains all the URLs referred during the startup phase.

The blacklist is used to detect abnormal API invoking. Each entry in the blacklist is a triplet, contains Javascript interface function name, JAVA interface function name, and sensitive API description. According to the developer document [6, 7] , a Javascript to JAVA interface is an one to one mapping, identified by using `@JavascriptInterface`. By decompile the captured apk files, we can collect all the Javascript interfaces. Then AdHoneyDroid generates the call graph of the app and analyze if the collected Javascript interface can invoke sensitive API calls. In this paper, sensitive API calls contains the APIs which should not be used in ad libraries such as `SmsManager.sendTextMessage` and `TelephonyManager.getLine1Number`. These functions relate to users privacy and should not be invoked by any ad libraries. So the entry triplet <A, B, C> means that Javascript interface A directly calls JAVA function B, and finally invokes sensitive API C. AdHoneyDroid generates this kind of blacklist entries as many as possible.

Running.

In the running step, AdHoneyDroid executes the collected app in an emulator and monitor all the sensitive API calling. On each sensitive API calling, AdHoneyDroid check the call stack and looks for the JAVA interface and Javascript interface in the blacklist. If a sensitive API call has a call stack which matches a blacklist entry, we believe that an malicious advertisement successfully triggered an attack and invokes sensitive API functions. What's more, AdHoneyDroid use TaintDroid to monitor privacy leakage attacks. On any sensitive information leakage through sensitive API calling, an attack is also reported.

Logging.

In the logging step, AdHoneyDroid uses the whitelist first to eliminate false positives. Then AdHoneyDroid stores the incoming malicious advertisement URL and content together with the triggered blacklist entry into database. This information is very useful for future analysis.

3. RELATED WORKS

As far as we know, there is no similar approach which can collect malicious advertisements. Most of similar works focus on security analysis of advertisement libraries such as [8–12]. Grace et al [8] reveals potential security threats of ad libraries by using static analysis. Stevens et al [9] analyzes the privacy risks in Android ad libraries. AdSplit [10], AdDroid [11] and SanAdBox [12] provide ad library security protection techniques. However, these works do not capture incoming malicious ads.

4. CONCLUSIONS

In this paper we propose the design and implementation of AdHoneyDroid, a malicious ad capturing honeypot system. AdHoneyDroid can automatically collect apps from markets and monitor their execution. By using static analysis, AdHoneyDroid generates blacklists for sensitive API calls. AdHoneyDroid use the blacklist together with Taintdroid to detect malicious advertisement attacks. AdHoneyDroid also generate whitelists to reduce false positive rates. We believe that AdHoneyDroid is a practical system which can be used in industry.

5. ACKNOWLEDGEMENT

This research is supported by the National Natural Science Foundation of China (Grant No. 61402025).

6. REFERENCES

[1] Israel Mojica Ruiz, Meiyappan Nagappan, Bram Adams, Thorsten Berger, Steffen Dienst, and Ahmed Hassan. On the relationship between the number of ad libraries in an android app and its rating. *IEEE Software*, 99(PrePrints):1, 2014.

[2] FireEye Inc. Js-binding-over-http vulnerability and javascript sidedoor: Security risks affecting billions of android app downloads. `http://goo.gl/eAFHEK`.

[3] Google Inc. Android platform version distribution. `http://developer.android.com/about/dashboards/index.html#Platform`.

[4] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. Taintdroid: An information flow tracking system for real-time privacy monitoring on smartphones. *Commun. ACM*, 57(3):99–106, March 2014.

[5] Google Inc. Google play. `https://play.google.com/store`.

[6] Google Inc. @javascriptinterface. `http://developer.android.com/guide/webapps/webview.html#BindingJavaScript`.

[7] Google Inc. addjavascriptinterface. `http://developer.android.com/reference/android/webkit/WebView.html#addJavascriptInterface(java.lang.Object,java.lang.String)`.

[8] Michael C. Grace, Wu Zhou, Xuxian Jiang, and Ahmad-Reza Sadeghi. Unsafe exposure analysis of mobile in-app advertisements. In *Proceedings of the Fifth ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WISEC '12, pages 101–112, New York, NY, USA, 2012. ACM.

[9] Ryan Stevens, Clint Gibler, Jon Crussell, Jeremy Erickson, and Hao Chen. Investigating user privacy in android ad libraries. Citeseer.

[10] Shashi Shekhar, Michael Dietz, and Dan S. Wallach. Adsplit: Separating smartphone advertising from applications. In *Proceedings of the 21st USENIX Conference on Security Symposium*, Security'12, pages 28–28, Berkeley, CA, USA, 2012. USENIX Association.

[11] Paul Pearce, Adrienne Porter Felt, Gabriel Nunez, and David Wagner. Addroid: Privilege separation for applications and advertisers in android. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '12, pages 71–72, New York, NY, USA, 2012. ACM.

[12] H. Kawabata, T. Isohara, K. Takemori, A Kubota, J. Kani, H. Agematsu, and M. Nishigaki. Sanadbox: Sandboxing third party advertising libraries in a mobile application. In *Communications (ICC), 2013 IEEE International Conference on*, pages 2150–2154, June 2013.