An empirical study of click fraud in mobile advertising networks

Geumhwan Cho, Junsung Cho, Youngbae Song, Hyoungshick Kim
Department of Computer Science and Engineering
Sungkyunkwan University, Republic of Korea
Email: {geumhwan, js.cho, youngbae, hyoung}@skku.edu

***Abstract*—Smartphone advertisement is increasingly used among many applications and allows developers to obtain revenue through in-app advertising. Our study aims at identifying potential security risks of a type of mobile advertisement where advertisers are charged for their advertisements only when a user clicks (or touches) on the advertisements in their applications. In the Android platform, we design an automated click generation attack and empirically evaluate eight popular advertising networks by performing real attacks on them. Our experimental results show that six advertising networks (75%) out of eight (`Millennial Media`, `AppLovin`, `AdFit`, `MdotM`, `RevMob` and `Cauly Ads`) are vulnerable to our attacks. We also discuss how to develop effective defense mechanisms to mitigate such automated click fraud attacks.**

***Index Terms*—advertising network; click fraud; Android;**

I. Introduction

As smartphones become more popular, the mobile advertisement market is also growing rapidly [4]. Mobile advertisement is a primary business model that offers the financial incentives for developers to distribute free applications. In this business model, advertising networks serve as a single vendor for advertisers and pay a developer according to the numbers of *impressions* (the number of times an advertisement has been served) and/or *clicks* generated by users [9]; application developers expect users to "pay" for their applications by viewing (i.e., impressions) or clicking advertisements (i.e., clicks) as many as possible.

One of the challenges in this mechanism is to detect and prevent fraudulent clicks, that have no intention of generating value [6]. Although this security issue has been extensively studied, most studies have focused on preventing click-fraud attempts housed on web pages rather than mobile platforms [2], [17].

In this paper, we particularly focus on the discussion of security risks associated with smartphone advertisement. In other words, we will show how vulnerable mobile advertising networks are to automatic attacks. Crussell et al. [1] analyzed the prevalence of fraudulent advertisement behaviors generated by real Android apps. We here extend their work by implementing an automated click generation tool called *ClickDroid* and empirically test its feasibility with real services. Unlike the previous study [1], which is based on emulation results, we apply our tool to the eight real advertising networks (`AdMob`, `Millennial Media`, `AppLovin`, `AdFit`, `MdotM`, `LeadBolt`, `RevMob` and `Cauly Ads`) and demonstrate that automatically generated clicks were successfully approved in the 6 advertising networks (out of 8 networks). We have summarized our contributions as follows.

- We implement a tool (*ClickDroid*) capable of automatically generating clicks for advertisements to simulate user clicks with dynamically changing device's identifier to bypass security policies (e.g., limiting the number of clicks each device produces per day) that are popularly used in advertising networks.
- We show the feasibility of automatic click generation attack on eight popular mobile advertising networks to evaluate their security risks. 75% of the systems that we experimented (`Millennial Media`, `AppLovin`, `AdFit`, `MdotM`, `RevMob` and `Cauly Ads`) did not detect our anomalous click attempts.
- We suggest reasonable countermeasures to detect and prevent automated fraudulent clicks, where client-side and server-side defense solutions can be deployed to mitigate those attacks.

The rest of this paper is organized as follows. In Section II, we provide some background on the mobile advertisement (particularly for the Android platform). Then we present how *ClickDroid* works through our prototype implementation in Section III. In Section IV, we identify the vulnerability of six real advertising networks by conducting experiments. In Section V, we discuss three practical defense mechanisms to detect and prevent automated fraudulent clicks. In Section VI, we explain how ethical issues were considered in the experiments. The related work is reviewed in Section VII. Our conclusions are in Section VIII.

II. Background

In this section, we explain definitions of the terminologies used in the remaining of the paper. We also explore the revenue models of the mobile advertising networks and how they work in Android to provide a better understanding of our attack mechanism.

A. Terminology

We define the following definitions.

- *Publisher* is an entity which deploys a mobile application with advertisements.

978-1-4673-6590-1/15 $31.00 © 2015 IEEE
DOI 10.1109/ARES.2015.62

- *Advertiser* is an entity which pays advertising networks for being displayed her advertisements on applications.
- *Advertising network (Ad network)* is an entity which manages publishers and advertisers. They can buy and sell advertisement traffic through trusted partner networks.
- *Impressions* are a metric on counting the number of times an advertisement has been deployed.
- *Clicks* are another metric on counting the number of events that are generated when users click on an advertisement.
- *Advertisement request (Ad request)* is the form of `HTTP` traffic that are generated from impressions or clicks. Whenever a valid request message is generated (i.e., advertisement is shown to a user or clicked by a user), advertisers have to pay the ad network and a percentage of amount is also paid to the publisher.

B. Revenue Model

Mobile advertisement services are dramatically growing up as the number of smartphone users is also increasing. In particular, the Android market offers the opportunity for advertisers and publishers who are interested in the mobile advertisement business. A publisher (i.e., application developer) releases a (free) mobile application with advertisements so that she can make money through those advertisements where revenue is typically determined by the amounts of *impressions* and/or *clicks*. The following revenue models are generally used:

- Cost per mile (**CPM**) is that advertisers charge per a thousand impressions to publishers with ad networks. It is also referred as the cost per thousand (CPT) since it has estimated the cost per thousand views of the advertisements. This measurement is widely available for advertisements for Android developers.
- Cost per click (**CPC**) is that advertisers charge per click which is generated by users. It is used for the number of times a website visitor or a user for an application clicks on a banner. This measurement is also popularly used since it can be implemented in a simple manner.
- Cost per action (**CPA**) is that advertisers charge per specific action such as filling a form, signing up for an offer, completing a survey, or downloading software. This model seems advantageous to advertisers since they only pay for specific actions which are directly related to their advertisements. However, it is not easy to implement when it comes to complex actions.

In this paper, we particularly focus on the discussion of security concerns raised by the CPC revenue model. We show that most ad networks (`Millennial Media`, `AppLovin`, `AdFit`, `MdotM`, `RevMob` and `Cauly Ads`) using the CPC revenue model can be vulnerable to automated click generation attacks.

C. How Ad Networks Work

As we have already explained in Section II-A, an ad network acts as a moderator between publishers and advertisers. In Android, a *jar* file acts as a moderator, which should be included in an application to embed advertisements into publisher's applications. When a publisher wants to display advertisements as a part of its application, she must sign up with the ad network and download the advertisement library. That library typically provides an API for embedding advertisements into the UI of the publisher's application and fetching, rendering and tracking advertisements. The device identifier is generally used to uniquely identify the publisher, who wanted to embed those advertisements.

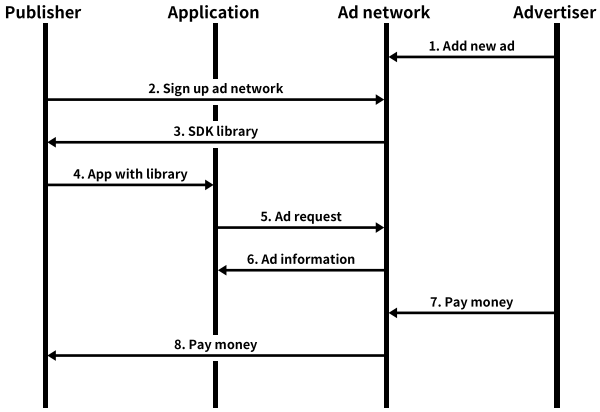


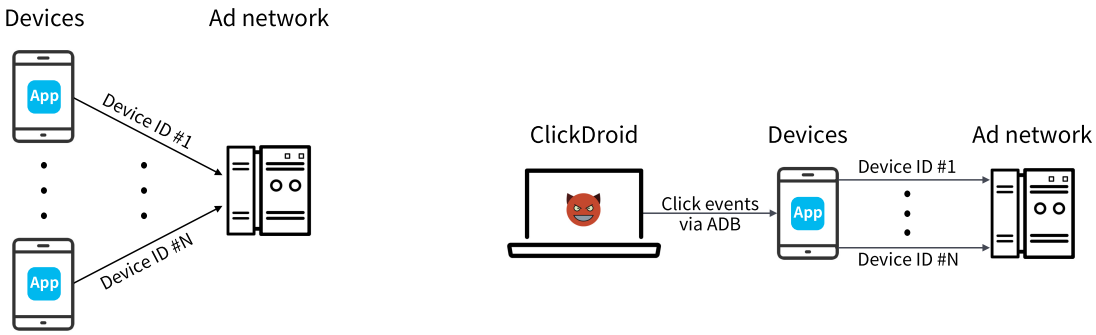
Fig. 1. How ad networks work to manage publishers, applications and advertisers with advertisement library

As shown in Figure 1, we illustrate how ad networks manage publishers, applications and advertisers with advertisement library. Since an advertiser A wishes to send advertisements to many Android users, she requests to distribute her advertisement via an ad network N. After receiving the request, the ad network adds the advertiser's advertisement to the ad network's SDK library. Imagine that there is a publisher P who wants to make money with mobile advertising services. When the publisher signs up with the ad network, she downloads the latest ad network's SDK library and release her application App with this library.

When a user clicks on the advertisement related to the advertiser A, the application App delivers the *ad request* message containing the user device's identifier to the ad network N. The ad network N then sends the response to the application App running on the user's device. For that click on the advertisement banner, the advertiser A pays the ad network N and the publisher P.

III. Implementation of ClickDroid

As we can see in Section II, in the CPC revenue model, a publisher makes money whenever a user clicks on the advertisement. Therefore a malicious publisher may involve in the act of generating such click events on the advertisement with the publisher's profit. In theory, those events can be simply generated. However, it is still questionable whether real advertisement networks are vulnerable to those attacks



(a) Normal situation with multiple devices

(b) Pretending to be normal by *ClickDroid*

Fig. 2. *ClickDroid* generates click events with different device identifiers, respectively, such that those look like click events generated by multiple devices.

since they may provide some countermeasures to defeat such attacks.

To analyze the risk of real mobile ad networks, we implemented a test app called *ClickDroid* that can automatically generate click events for advertisements displayed on Android apps. We note that a malicious publisher can implement such an app; and the publisher can also install the app on his (or her) own device as `root` to generate fake clicks within a short time interval in order to make its own profit.

For simplicity, we used `Android Debug Bridge` (ADB) to implement *ClickDroid*, which is a debug support tool to communicate between a host and an Android device. In other words, with ADB, we can control an Android device without any restrictions. To generate a click event on an advertisement embedded in our prototype with a victim ad network's SDK library, *ClickDroid* sends a `sendevent` command to the Android app. Finally, a virtual click event is generated on an advertisement in the Android app. We note that *ClickDroid* can also be implemented as a standalone Android app without ADB support.

However, the generation of click events alone is not enough. We empirically found that some ad networks (e.g., `Cauly Ads`) have checked the requesting device's identifiers such as `IMEI` (International Mobile Equipment Identity) and/or "`Device ID`" (also known as `android_id`) to limit the number of possible ad requests from a device during a specific period (e.g., a day). In our empirical experiments, `Cauly Ads` limited the number of click events to 19 per day for a device (based on the "`Device ID`" information). If a device generates click events more than the threshold number (e.g., 19 per day in `Cauly Ads`), the device might be blacklisted by an ad network. Therefore, we tried to generate click events which resembles events generated by multiple devices associated with multiple users. Figure 2 illustrates how *ClickDroid* pretends to be a normal one with multiple devices. *ClickDroid* can generate not only click events but also (faked) *device identifiers* to disable device detection for limiting the number of ad requests from a single device.

A. Generation of Device IDs

In Android, "`Device ID`" (or `android_id`) is a 64-bit number (as a hex string) that is randomly generated when the user first sets up the device and should remain constant for the lifetime of the Android device[1]. This information can be used for identifying or tracking a device (particularly for tablet devices that do not have `IMEI`). However, we can observe that "`Device ID`" is independently (and randomly) created from an Android device itself. Therefore we can generate new `Device IDs` for an Android device without any restrictions.

Our main concern here is how to replace the existing "`Device ID`" with newly generated ones. The simplest solution is to perform the "factory" reset which is a procedure that securely erases all user data on the Android device and returns the device to its initial state. However, the whole process is time consuming; an Android app with the victim's SDK library should be installed again even when the Android device is successfully initialized.

Alternatively, we can update the "`Device ID`" value without performing the "factory" reset. Android saves all device settings including "`Device ID`" in a `SQLite` database file[2]. This database file can be managed by a program named `sqlite3` at runtime. We assume that `sqlite3` is already installed on the Android device. *ClickDroid* can open the `settings.db` file and modify the "`Device ID`" value via ADB (see Figure 3) before generating ad requests.

B. Generation of Ad Requests

In order to successfully send ad requests to an ad network, *ClickDroid* sequentially generates a series of pre-defined click events via ADB: (1) *ClickDroid* first starts to run an Android app with the victim's SDK library, (2) generates a click event on the advertisement in the app, and then (3) updates "`Device ID`" with a new random 64-bit number. When `Device ID` is successfully changed, *ClickDroid* repeats this

[1]http://developer.android.com/reference/android/provider/Settings.Secure.html

[2]/data/data/com.android.providers.settings/databases/settings.db

```
$ sqlite3 /data/data/com.android.providers.settings/databases/settings.db

sqlite> update secure set value='ce8946b96e22354e' where name='android_id';
```

Fig. 3. Example of `SQLite` commands to update "`Device ID`" (also known as `android_id`).

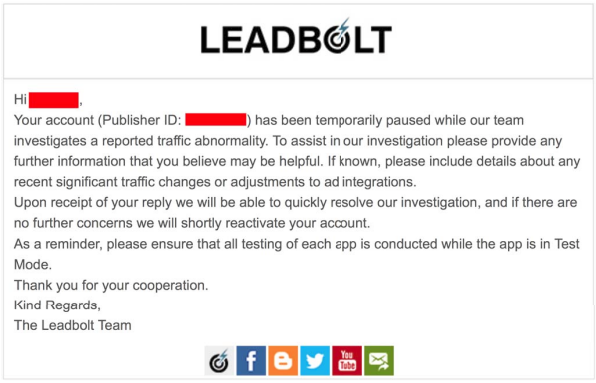


Fig. 4. Our `LeadBolt` account was temporarily paused due to traffic abnormality.

procedure from (2). The reason we use a new `Device ID` for every request is to avoid detection by the ad network. That is, when the ad network monitors ad request traffic to detect click fraud by counting the number of ad requests from the same device, *ClickDroid* can trick the ad network with new `Device IDs` into believing as if those requests came from different users (or Android devices) individually.

IV. Experiments

ClickDroid (which can be assumed as the attacker's malicious app) was used for evaluating the security risk of real ad networks. We perform click generation attacks on eight popular ad networks (`AdMob`, `Millennial Media`, `AppLovin`, `AdFit`, `MdotM`, `LeadBolt`, `RevMob` and `Cauly Ads`), respectively. Those ad networks were selected by "AppBrain" website that offers the lists of top 500 most installed ad networks[3]. We selected those ad networks which are suitable for testing the CPC revenue model with a banner. For each ad network, we create a user account to receive payments and implement an Android app with the SDK library for advertisements of the ad network. We note that a malicious publisher (as also the app developer) uses his (or her) own device in order to use *ClickDroid* on the device without any restriction.

In our threat model, the attacker's (i.e., publisher's) goal is to successfully deliver ad request messages to a victim ad network to receive payments for those messages from the ad network. Therefore, we tested whether ad request messages can be successfully delivered whenever *ClickDroid* attempts to generate a click (with a different `Device ID`) on the victim's advertisements and our account is still valid. Specifically, we assume that the attack is successful if *ClickDroid* generates over 100 ad request messages without any disruption to receive payments for the generated clicks.

In our experiments, 75% (`Millennial Media`, `AppLovin`, `AdFit`, `MdotM`, `RevMob` and `Cauly Ads`) of the ad networks that we analyzed failed to prevent the automated click generation attacks conducted by *ClickDroid*. Probably, this implies that many real-world ad networks seem to be significantly dangerous due to click fraud. In this paper, our attack attempts are not sophisticated but straightforward. However, 75% of the ad networks are vulnerable to those attempts.

In this paper, we claim that many ad networks are not actually be interested in mitigating click fraud. Perhaps this is another example of "negative externality" [12] in the field of information security, which is an economic activity that imposes a negative effect on an unrelated third party. In fact, click fraud may incur significant losses in *advertisers* instead of *ad networks*.

Fortunately, the other two mobile ad networks (`AdMob` and `LeadBolt`) are secure against automated click generation attacks. When *ClickDroid* generates clicks even with uniquely different `Device IDs`, those networks detect our attacks only with a small number of attack attempts and then finally blocked our accounts. For example, our account used in the experiments for `LeadBolt` was temporarily paused due to traffic abnormality (see Figure 4). We surmise that those networks might have their own defense mechanisms to detect such abnormal request patterns. We will discuss such defense mechanisms in detail in the next section.

V. Countermeasures

In this section, we describe several mechanisms possible for the prevention of *ClickDroid* (i.e., automatic click generation on Android's banner advertisements) attacks.

A. Distinguishing Physical Clicks from Software Generated Clicks

To prevent *clicks* generated by malware such as *ClickDroid*, we might try to distinguish touch events *physically* generated by a human user from those virtually generated by software. The main idea is to collect a series of touch events (where each touch event includes its XY-coordinate values and generation time) physically generated by a human user from the touch screen device driver and deliver them to the `InputDispatcher` at Android framework in order to

[3]http://www.appbrain.com/stats/libraries/ad?list=top500

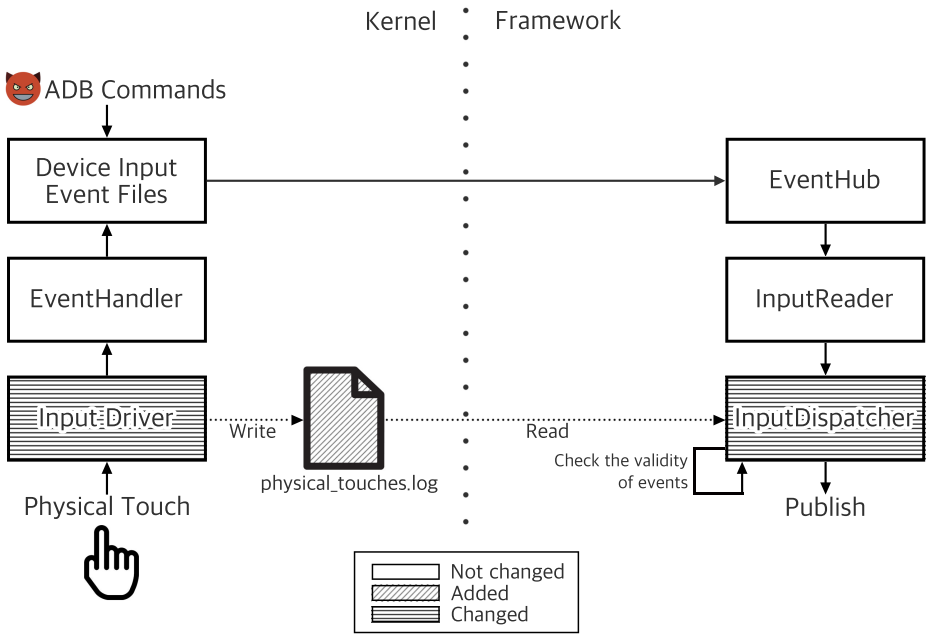


Fig. 5. Phisical Touch Logging Model

detect artificially generated touch events by software before processing them for Android apps.

We propose a modification of the existing Android architecture to trace physically generated touch events on the device screen (see Figure 5). Android builds on the top of a Linux kernel and includes a middleware framework and an application layer. When a human user touches a region on the physical screen, the touch event can be collected through the function `input_event_from_user` at the kernel level. In our modification, this event information (the coordinate of the touch and its time-stamp) is stored to a file named `physical_touches.log`. In contrast, the touch events artificially generated via ADB are not stored in that file. Based on this difference between human and software generated events, we can effectively detect click events generated by *ClickDroid* before actually handling them in the `InputDispatcher` at the framework layer.

When a new touch event arrives, the proposed system checks whether this event information is correctly recorded in the file of `physical_touches.log` before publishing the touch event. If there exists the event information in `physical_touches.log`, that event is successfully published through the `InputDispatcher` and the event information is removed from `physical_touches.log`; otherwise, that event is properly processed according to a predefined security policy (e.g., "just ignore such events").

B. Honey Advertisement

'Honey' is the traditional term used to indicate a 'decoy' or 'bait' for attackers in the field of security. For example, a honeypot is a security resource, which is intended to be attacked and compromised to gain more information about an attacker and his attack techniques [16].

To mitigate click fraud, we suggest an advertising system for automatic click generation software such as *ClickDroid*. We call this approach "honey advertisement". Unlike the existing advertisement systems, ad networks' SDKs often display *transparent* advertisement banners (in a random manner) to deceive malicious programs that automatically generate click events on those banners. A human cannot see those banners while a machine cannot distinguish transparent banners from normal banners. Therefore if an ad request for a transparent banner was generated, this request might be triggered by automatic click generation software rather than a human user and can finally be used to set off an alarm of an automatic click fraud attack on the advertising infrastructure. This technique was introduced in the previous work [5].

C. Detecting Anomalous Behaviors

We can see that ad requests generated by *ClickDroid* have significantly different patterns compared with those generated by human users — for example, the automatically generated requests would be periodically repeated during a relatively short time. Oentaryo et al. [13] and Kitts et al. [7] introduced systems, respectively, to detect click fraud patterns in online advertisement using server-side event models. Since the existing Android platform can be used without modifications to support this approach, we highly recommend deploying similar systems at server-side.

As presented in Section IV, we believe that some of ad networks (e.g., `AdMob` and `LeadBolt`) might already use

such systems to detect abnormal request patterns for click fraud in mobile platforms.

VI. Ethical Considerations

The main motivation of our experiments is to analyze potential risks of mobile advertisement services and suggest reasonable countermeasures to mitigate such risks. Therefore we only checked ad networks' responses for our click fraud attempts; however, actual money is not withdrawn from our bank accounts. We also reported the discovered design flaws to the ad networks, which acknowledged them.

Another ethical concern is related to "`Device ID`". When we replace "`Device ID`", the used `Device ID` may belong to some legitimate user that might be potentially blamed for our attack experiments. However, we note that this possibility is very unlikely because the used "`Device IDs`" are randomly generated ones rather than real Android devices' "`Device ID`".

VII. Related Work

Over the last few years, online advertisement has been widely studied because it has become a significant source of revenue for web-based businesses. However, it also introduces a new type of cyber criminal activities called "click fraud". Click fraud is the practice of deceptively clicking on advertisements with the intention of either increasing third-party website revenues or exhausting an advertiser's budget [18]. Kshetri [8] examined the mechanisms and processes associated with the click-fraud industry from an economics viewpoint. Miller et al. [11] analyzed the characteristics of real-world click fraud by examining the operations and underlying economic models of two modern malware families, `Fiesta` and `7cy`, which are typically used for click fraud.

To prevent those click frauds, several defense techniques have been introduced. The simplest solution is to use threshold-based detection. If a website is receiving a high number of click events with the same device identifier (e.g., IP address) in a short time interval, those events can be considered as fraud. However, click fraud detection is not trivial — clicks can sophisticatedly be generated to bypass such naive defense schemes. For example, attackers are behind proxies or globally distributed [5]. Also, device identifiers such as IP address can easily be modified.

To mitigate such sophisticated attacks, Kitts et al. [7] discussed how to design a data mining system to detect large-scale click fraud attacks. Metwally et al. [10] developed a technique based on the traffic similarity analysis to discover a type of fraud called *coalitions* performed by multiple fraudsters. Another interesting approach is to use bait advertisements [3], [5]. Xu et al. [19] proposed a systematic approach by introducing additional tests to check whether visiting clients are clickbots.

Only a few studies have analyzed the security of mobile advertising networks although many applications use one or more advertising services as a source of revenue for the Android developers. Those studies were mainly focused on discussing the security concerns about unnecessary permissions required by advertisement libraries. Pearce et al. [14] showed that 49% of Android applications contain at least one advertisement library, and these libraries overprivilege 46% of advertising-supported applications. Shekhar et al. [15] proposed an approach called *AdSplit* to separate applications from its advertisement libraries that might request permissions for sensitive privileges.

When it comes to click fraud in mobile platforms, Crussell et al. [1] raised the issue about click fraud in the context of mobile advertising. However, their study results may not be sufficient to show the real impacts of click fraud attacks in mobile platforms because their study mainly focused on analyzing existing mobile applications' fraudulent behaviors that could be used for advertisement fraud. In contrast, we studied how vulnerable real mobile advertising networks are to click fraud attacks by performing real attacks on them.

VIII. Conclusion

This paper evaluates the risk of automated click generation attacks on mobile ad networks. Our experimental results show that 75% of those networks (`Millennial Media`, `AppLovin`, `AdFit`, `MdotM`, `RevMob` and `Cauly Ads`) are vulnerable to our attacks. However, our current results are not enough to generalize our observations because of the limited number of tested ad networks. Therefore, as an extension to this paper, we need to consider performing our tests on a large sample of ad networks.

To mitigate such automated attacks, we suggest three reasonable defense mechanisms. In future work, we plan to implement those mechanisms and evaluate their performance against the attacks. As another extension to this work, we also plan to conduct intensive experiments with other mobile ad networks to generalize our observations.

Acknowledgment

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (No. 2014R1A1A1003707). This research was also supported by the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2015-H8501-15-1008) supervised by the IITP (Institute for Information & communications Technology Promotion), and funded in part by the ICT R&D program (2014-044-072-003 , 'Development of Cyber Quarantine System using SDN Techniques') of MSIP/IITP.

Authors would like to thank Priya Anand for proofreading the paper, and all the anonymous reviewers for their valuable feedback.

References

[1] J. Crussell, R. Stevens, and H. Chen. Madfraud: Investigating ad fraud in android applications. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, pages 123–134. ACM, 2014.

[2] N. Daswani, C. Mysen, V. Rao, S. Weis, K. Gharachorloo, and S. Ghosemajumder. Online advertising fraud. *Crimeware: Understanding New Attacks and Defenses*, 40(2):1–28, 2008.

[3] V. Dave, S. Guha, and Y. Zhang. Measuring and Fingerprinting Click-spam in Ad Networks. *ACM SIGCOMM Computer Communication Review*, 42(4):175–186, 2012.

[4] S. Dhar and U. Varshney. Challenges and business models for mobile location-based services and advertising. *Communications of the ACM*, 54(5):121–128, 2011.

[5] H. Haddadi. Fighting Online Click-Fraud Using Bluff Ads. *ACM SIGCOMM Computer Communication Review*, 40(2):21–25, 2010.

[6] N. Immorlica, K. Jain, M. Mahdian, and K. Talwar. Click fraud resistant methods for learning click-through rates. In *Proceedings of the First International Workshop on Internet and Network Economics*, pages 34–45. Springer, 2005.

[7] B. Kitts, J. Zhang, G. Wu, W. Brandi, J. Beasley, K. Morrill, J. Ettedgui, S. Siddhartha, H. Yuan, F. Gao, et al. Click fraud detection: Adversarial pattern recognition over 5 years at microsoft, 2013.

[8] N. Kshetri. The economics of click fraud. *Security Privacy, IEEE*, 8(3):45–53, May 2010.

[9] I. Leontiadis, C. Efstratiou, M. Picone, and C. Mascolo. Don't kill my ads!: Balancing privacy in an ad-supported mobile application market. In *Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications*, pages 2:1–2:6, 2012.

[10] A. Metwally, D. Agrawal, and A. El Abbadi. Detectives: Detecting Coalition Hit Inflation Attacks in Advertising Networks Streams. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 241–250, 2007.

[11] B. Miller, P. Pearce, C. Grier, C. Kreibich, and V. Paxson. Whats clicking what? techniques and innovations of todays clickbots. In *Proceedings of Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 164–183. Springer, 2011.

[12] T. Moore, R. Clayton, and R. Anderson. The economics of online crime. *The Journal of Economic Perspectives*, 23(3):3–20, 2009.

[13] R. Oentaryo, E.-P. Lim, M. Finegold, D. Lo, F. Zhu, C. Phua, E.-Y. Cheu, G.-E. Yap, K. Sim, M. N. Nguyen, et al. Detecting click fraud in online advertising: a data mining approach. *The Journal of Machine Learning Research*, 15(1):99–140, 2014.

[14] P. Pearce, A. P. Felt, G. Nunez, and D. 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, 2012.

[15] S. Shekhar, M. Dietz, and D. S. Wallach. Adsplit: Separating smartphone advertising from applications. In *Proceedings of USENIX Security Symposium*, pages 553–567. USENIX, 2012.

[16] L. Spitzner. *Honeypots: Tracking Hackers*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[17] B. Stone-Gross, R. Stevens, A. Zarras, R. Kemmerer, C. Kruegel, and G. Vigna. Understanding fraudulent activities in online ad exchanges. In *Proceedings of the 2011 ACM Conference on Internet Measurement Conference*, pages 279–294. ACM, 2011.

[18] K. C. Wilbur and Y. Zhu. Click Fraud. *Marketing Science*, 28(2):293–308, 2009.

[19] H. Xu, D. Liu, A. Koehl, H. Wang, and A. Stavrou. Click fraud detection on the advertiser side. In *Proceedings of the 19th European Symposium on Research in Computer Security*, pages 419–438. Springer, 2014.