RESEARCH   **Open Access**

CrossMark

Combating online fraud attacks in mobile-based advertising

Geumhwan Cho[1], Junsung Cho[1], Youngbae Song[1], Donghyun Choi[2] and Hyoungshick Kim[1*]

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 mobile-based advertising services where advertisers are charged for their advertisements on mobile applications. In the Android platform, we particularly implement bot programs that can massively generate click events on advertisements on mobile applications and test their feasibility with eight popular advertising networks. Our experimental results show that six advertising networks (75 %) out of eight are vulnerable to our attacks. To mitigate click fraud attacks, we suggest three possible defense mechanisms: (1) filtering out program-generated touch events; (2) identifying click fraud attacks with faked advertisement banners; and (3) detecting anomalous behaviors generated by click fraud attacks. We also discuss why few companies were only willing to deploy such defense mechanisms by examining economic misincentives on the mobile advertising industry.

Keywords: Advertising network, Click fraud, Android

1 Introduction

As smartphones become more popular, the mobile advertisement market is also growing rapidly [1]. Mobile advertisement is a primary business model that offers the financial incentives for developers to distribute free applications. In the mobile advertisement market, 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 [2]; application developers expect users to "pay" for their applications by viewing (i.e., impressions) or clicking advertisements (i.e., generating clicks) as many as possible.

In those business models, the most important security problem is to detect and prevent (artificially created) fraudulent events, which have no intention of generating value for advertising [3]. 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 [4, 5].

Recently, there have been a few attempts to analyze the security risks of smartphone advertisement. Crussell et al. [6] particularly analyzed the prevalence of fraudulent advertisement behaviors generated by real Android apps.

In this paper, we extend their work by implementing independent bot programs to generate fraudulent click events in an automatic manner. Unlike the previous study [6], which is based on emulation results, we applied our bot programs to the eight real advertising networks (`AdMob`, `Millennial Media`, `AppLovin`, `AdFit`, `MdotM`, `LeadBolt`, `RevMob`, and `Cauly Ads`) and found that artificially generated click events were successfully approved in the six advertising networks (out of eight networks). We highlight our key contributions as follows:

- We design and develop bot programs capable of automatically generating fraud events to mimic users' activities on advertisements.
- We particularly show the feasibility of automatic click generation attacks on eight popular mobile advertising networks to evaluate their security risks. Seventy-five percent of the systems that we experimented (`Millennial Media`, `AppLovin`,

*Correspondence: hyoung@skku.edu
[1] Department of Computer Science and Engineering, Sungkyunkwan University, Seobu-ro 2066, 16419 Suwon, Republic of Korea
Full list of author information is available at the end of the article

`AdFit`, `MdotM`, `RevMob`, and `Cauly Ads`) did not detect our anomalous click attempts.

- We suggest three possible defense mechanisms: (1) filtering out program-generated touch events (at client side); (2) identifying click fraud attacks with faked advertisement banners (at client side); and (3) detecting anomalous behaviors generated by click fraud attacks (at server side).
- We discuss the issue of economic misincentives on the mobile advertising industry to discover an inherent problem in using countermeasures against click fraud attacks.

The rest of this paper is organized as follows. The related work is reviewed in Section 2. In Section 3, we provide some background on the mobile advertisement (particularly for the Android platform). In Section 4, we present how bot programs can be implemented for online fraud in mobile-based advertising. Then, we evaluate the feasibility of bot programs against click-based advertisements through intensive experiments with real advertisement services in Section 5. In Section 6, we discuss three practical defense mechanisms to detect and prevent automated fraudulent clicks. In Section 7, we explore the misincentive problem that can inherently corrupt ad networks through false clicking. Our conclusions are in Section 8.

2   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 [7]. Kshetri [8] examined the mechanisms and processes associated with the click fraud industry from an economics viewpoint. Miller et al. [9] 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 sophisticately be generated to bypass such naive defense schemes. For example, attackers are behind proxies or globally distributed [10]. Also, device identifiers such as IP address can easily be modified.

To mitigate such sophisticated attacks, Kitts et al. [11] discussed how to design a data mining system to detect large-scale click fraud attacks. Metwally et al. [12] 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 [10, 13]. Xu et al. [14] 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. [15] showed that 49 % of Android applications contain at least one advertisement library, and these libraries overprivilege 46 % of advertising-supported applications. Shekhar et al. [16] 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. [6] 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 implementing bot programs and testing their feasibility with real advertisement networks. We particularly extend our preliminary work [17] to generalize click fraud attacks with various revenue models and develop practical defense mechanisms for mitigating click fraud attacks on mobile devices. We also discuss the economic aspects of security failure that might be an inherent problem of click fraud in mobile advertising.

3   Background

In this section, we explain definitions of the terminologies used in the remaining of the paper. To provide a better understanding of click fraud attacks, we present how ad networks typically work between entities and explore business models popularly used for mobile advertising networks.

3.1   Terminology

We define the following definitions.

- *Publisher* is an entity which deploys a mobile application with advertisements.
- *Advertiser* is an entity which pays the advertising networks for their advertisements being displayed on applications.

Cho *et al. EURASIP Journal on Information Security* (2016) 2016:2

Page 3 of 9

- *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 is 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.

3.2 How ad networks work

As we have already explained in Section 3.1, 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 the 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.

As shown in Fig. 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

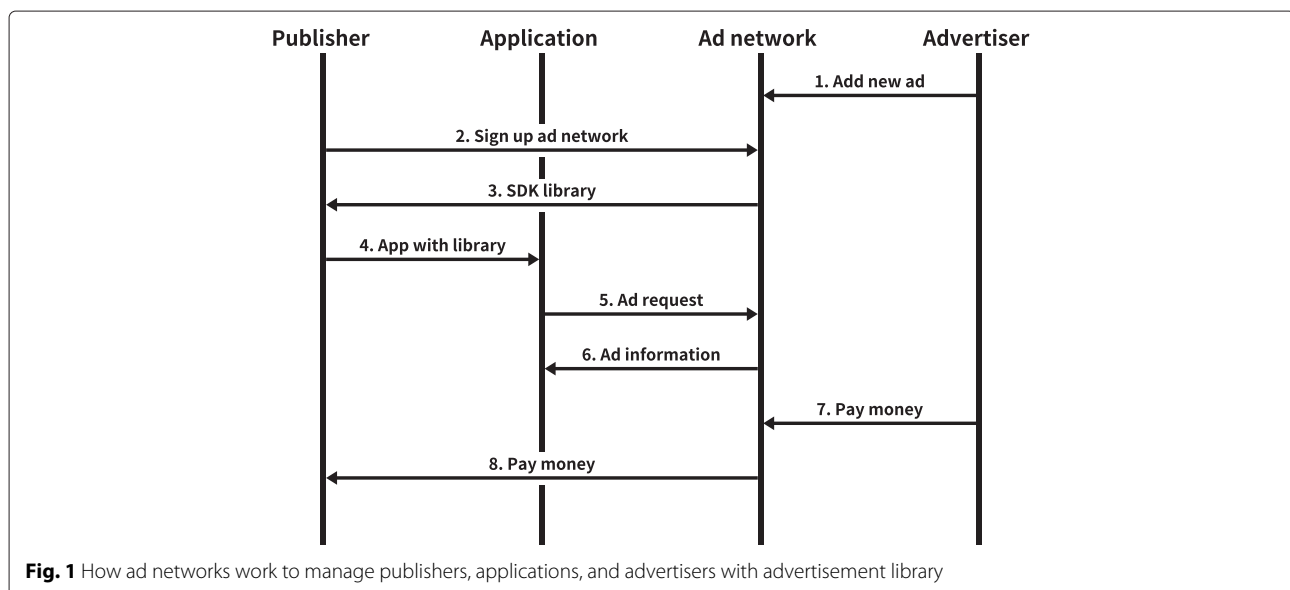distribute her advertisement via an ad network *N*. After receiving the request, the ad network adds the advertiser's advertisement to the ad network's software development kit (SDK) library. Imagine that there is a publisher *P* who wants to make money with mobile advertising services. When the publisher 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*.

3.3 Revenue models

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 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.

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

Cho *et al. EURASIP Journal on Information Security* (2016) 2016:2

Page 4 of 9

- 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 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.

4 Implementation of online fraud attacks

In this paper, we particularly focus on the implementation of bot programs against the CPC revenue model in order to test their feasibility with real ad networks (see Section 4.1). The test results were presented in Section 5. However, similar automatic attacks can also be implemented for the other revenue models. We will briefly introduce such attack designs in Section 4.2 and 4.3.

4.1 Attack against CPC

As we can see in Section 3, 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 since they may provide some countermeasures to defeat such attacks.

To analyze the risk of real mobile ad networks, we implemented an independent Android app that can automatically generate click events for advertisements displayed on Android apps. We note that a malicious publisher can implement such an app (i.e., bot program), and the publisher can also install the app on his (or her) own device to generate fake click events within a short time interval in order to make its own profit. This is a very effective economic activity for attackers. For example, in the case of `Cauly`, a publisher receives only US\$0.02 per click event—if an attack sequence is generated within 7 s on average, the attacker can earn US\$1480.72 in a week by running a bot program on the attacker's device.

For simplicity, we used `Android Debug Bridge` (ADB) 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, we send a `sendevent` command to the Android app via ADB. Finally, a virtual click event is generated on an advertisement in the Android app. Surely, such a bot program can also be implemented as a stand-alone 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 international mobile equipment identity (IMEI) 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
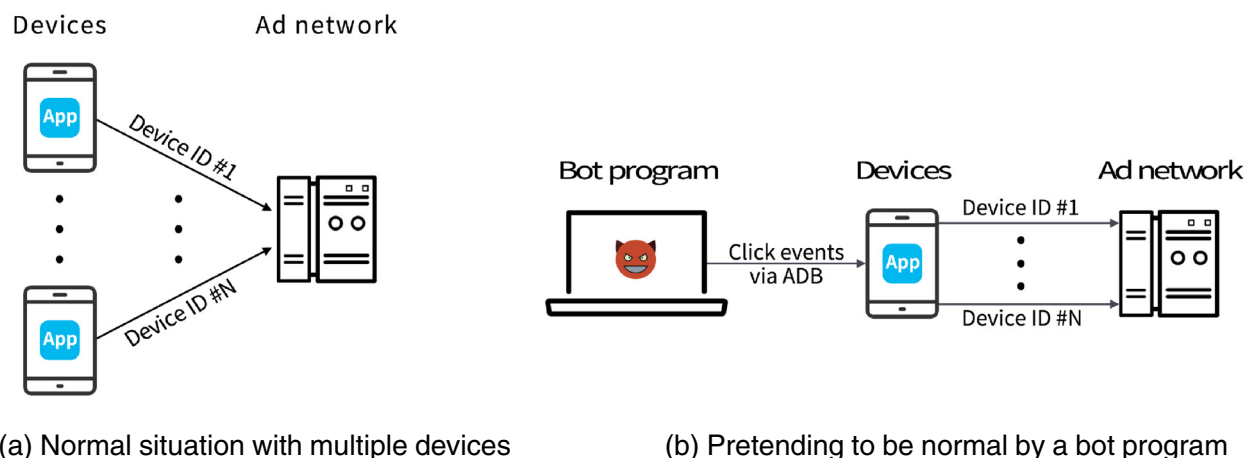

(a) Normal situation with multiple devices   (b) Pretending to be normal by a bot program

Fig. 2 Bot programs can generate click events with different device identifiers, respectively, such that those look like click events generated by multiple devices

how a bot program pretends to be a normal one with multiple devices. Bot programs 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.

4.1.1 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. A bot program can open the `settings.db` file and modify the `Device ID` value via ADB (see Fig. 3) before generating ad requests.

4.1.2 Generation of ad requests

In order to successfully send ad requests to an ad network, a bot program sequentially generates a series of pre-defined click events via ADB: (1) the bot program first starts to run an Android app with the victim's SDK library, (2) generates a series of click events on the advertisement in the app, and then (3) updates `Device ID` with a new random 64-bit number. Whenever `Device ID` is successfully changed, the bot program repeats this 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, the bot program can trick the ad network with new `Device IDs` into believing as if those requests came from different users (or Android devices) individually.

4.2 Attack against CPM

To automatically increase the number of impressions on a mobile application, the most straightforward approach is to view new advertisements as many as possible.

A typical ad network's SDK library updates its advertisements periodically (e.g., every 30 s). To reduce this delay time for updating advertisements, an attacker can make a targeted app with ad library terminate and restart in an automatic manner. In fact, a similar technique can also be needed for attacking CPC when multiple clicks are not allowed on the same advertisement.

However, since advertisers generally charge per thousand impressions, the effectiveness of such attacks is not comparable with the attacks against CPC in terms of efficiency. For example, in the case of `Cauly`, a publisher receives only $0.09 per thousand impressions.

4.3 Attack against CPA

The CPA model is a generalization of CPC; advertisers can consider various activities (e.g., watching a video clip, installing another app, signing up for a website) including click events.

In general, it is necessary to learn prior knowledge about specific event sequences for CPA activities in order to automatically generate those sequences against CPA. Surely, it seems to be trickier than implementing the attacks for CPC consisting of touch events with fixed screen positions.

5 Experiments

We implemented several independent Android apps with real ad networks' SDK library for evaluating their potential security risk against click fraud attacks. We performed click generation attacks described in Section 4 on eight popular ad networks (AdMob, Millennial Media, AppLovin, AdFit, MdotM, LeadBolt, RevMob, and Cauly Ads), respectively. Those ad networks were

```
$ 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`)

Cho *et al. EURASIP Journal on Information Security* (2016) 2016:2

Page 6 of 9

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 created a user account to receive payments and implemented 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 those apps 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 without any trouble whenever we had attempted to generate a click event (with a different `Device ID`) on the victim's advertisements. Specifically, we assume that the attack is successful if our bot program 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 bot programs. 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.

Fortunately, the other two mobile ad networks (`AdMob` and `LeadBolt`) are secure against automated click generation attacks. When we generate click events 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 Fig. 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.

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`.

6 Countermeasures
In this section, we describe several defense mechanisms for preventing click fraud (i.e., automatic click generation on Android's banner advertisements) attacks.

We extend our previous work [17] with more detailed implementation designs by fixing incorrect representation and clarifying some ambiguous parts in the previously suggested models.

6.1 Distinguishing human-generated touch events from program-generated touch events
To prevent touch events generated by bot programs for click fraud attacks, the most straightforward defense
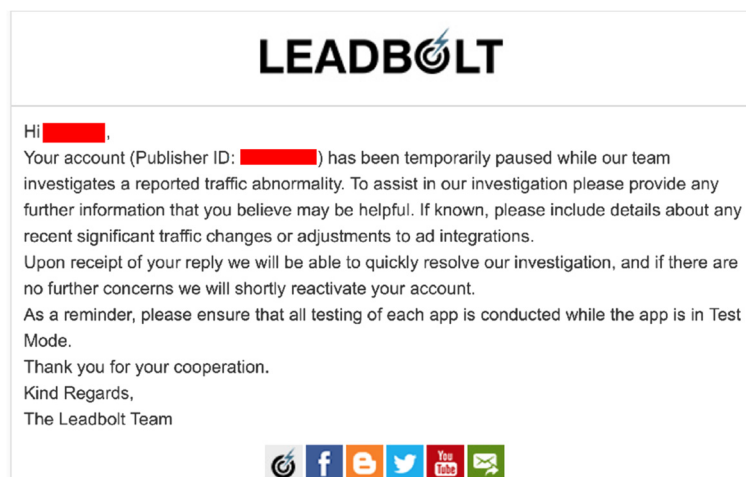

Fig. 4 Our `LeadBolt` account was temporarily paused due to traffic abnormality

Cho *et al. EURASIP Journal on Information Security* (2016) 2016:2

Page 7 of 9

mechanism is to effectively distinguish such events from human-generated touch events and filter out them. To achieve this goal, we first analyze how Android handles with touch events and then suggest a possible reference implementation to filter out program-generated touch events according to security policies.

We propose a modification of the existing Android architecture to trace physically generated touch events on the device screen (see Fig. 5). Android is built on the top of a Linux kernel and includes a middleware framework and an application layer. Touch events are gathered at the `/dev/input/event#` node called `Device Input Event` files regardless of either human-generated or program-generated events. Next, the `InputReader` framework reads those events from `Device Input Event` files and the collected events are dispatched by `InputDispatcher` to a proper app.

Before touch events are recorded at the `Device Input Event` files, human-generated touch events are processed through `Input Driver` while program-generated touch events are handled with the `evdev_write` function in the `evdev.c` file. We can develop a filter for program-generated events with this difference— the `evdev_write` function should be modified. We can accept or reject program-generated touch events before writing them on `Device Input Event` files depending on the target program's security policy defined in its `Security Policy` files. For example, program-generated touch events on a specific rectangle region (for displaying advertisements) can be ignored during the target program is actively running. We suggest that those files can be included in the ad network's SDK library and installed with the program itself together. We expect that

the existing architecture can be slightly modified to filter out automatically generated touch events by programs without significant loss in efficiency.

For using this filtering mechanism at the kernel level, however, the integrity of the `evdev_write` function and security policy files should be protected. In the legacy Android architecture, if a bot program with root privileges can be installed, it is not possible to satisfy this requirement because those files might be easily tampered by the bot program. Therefore, secure environments for the suggested technique should also be provided to protect those files from a malicious root. Hardware-assisted solutions (e.g., TrustZone-based Real-time Kernel Protection [18]) might be used to achieve this security goal.

We implemented a prototype of our defense framework to filter out program-generated touch events and tested its effectiveness against click fraud attacks on real mobile devices. We modified the Android operating system 5.1.1 (i.e., the `evdev_write` function). To simplify implementation, we used a naive security policy to ignore all touch events generated by programs. We tested its feasibility with a `Nexus 5` smartphone running the modified operating system against our own CPC bot implementation and a popular automatic click event generation tool called DummySprite (http://www.dummysprite.com) on Android. Our prototype implementation successfully prevented their attack attempts without incurring significant performance degradation.

6.2 Honey advertisement
"Honey" is the traditional term used to indicate a "decoy" or "bait" for attackers in the field of security. For example,
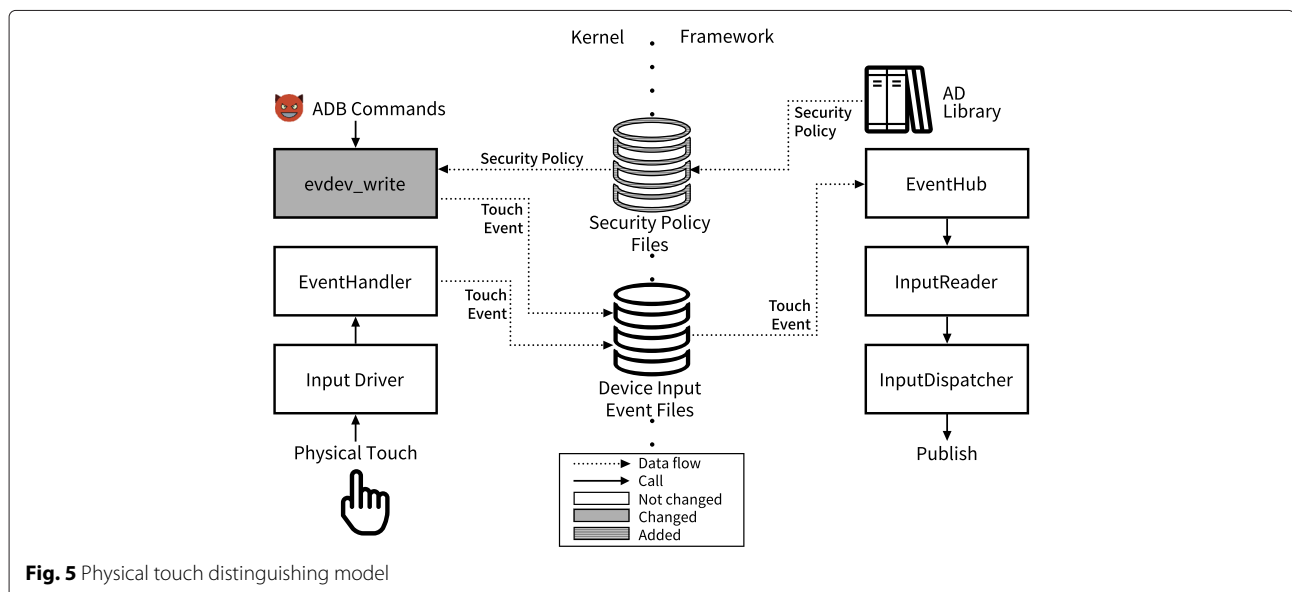

Fig. 5 Physical touch distinguishing model

Cho *et al. EURASIP Journal on Information Security* (2016) 2016:2

Page 8 of 9

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 [19].

To mitigate click fraud, we suggest an advertising system for automatic click generation software. 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 bot 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. A similar idea was introduced in the previous work [10].

We implemented a proof-of-concept app to show the feasibility of honey advertisement (see on Fig. 6). As shown in this figure, some advertisement banners can be transparently displayed in a random manner. In a normal situation, a visible banner image file with a link to the advertiser's server is displayed while a transparent image file is used with a link to the ad network's server in honey advertisement. When a transparent advertisement is clicked, the ad request is delivered to the ad network's server. Basically, such events are likely to be triggered by a bot program rather than a human user because human users cannot see transparent advertisements. Therefore, we may detect the bot program's existence with a low chance of false alarms.

6.3 Detecting anomalous behaviors

We can see that ad requests generated by bot programs 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. [20] and

Kitts et al. [11] 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 the server side.

As presented in Section 5, 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.

7 Economic aspects of security failure

In this section, we discuss why a small number of ad networks have only detected our straightforward click fraud attacks.

This might be explained from the economic aspects of security failure. When we examine the incentives of market players for mobile advertising, we might discover an inherent problem of click fraud.

As discussed in Section 6, several defense mechanisms fighting against online click fraud can be deployed. But, who should pay for those mechanisms? We note that click fraud may directly incur significant losses in advertisers instead of ad networks.

In theory, those solutions could be deployed by ad networks who are running mobile advertising platforms. However, we claim that many ad networks might not actually be interested in mitigating click fraud.

In the mobile advertising industry, ad networks manage publishers and advertisers as a moderator (see Section 3). In general, advertisers pay money to both ad networks and publishers for their advertisements. For example, in the CPC revenue model, whenever advertisements on mobile applications are clicked, publishers receives money; ad networks also earn money. That is, ad networks would rather profit from click fraud attacks than from defenses to mitigate such attacks.

With those misincentives between players, ad networks are not motivated enough to detect online fraud attacks.
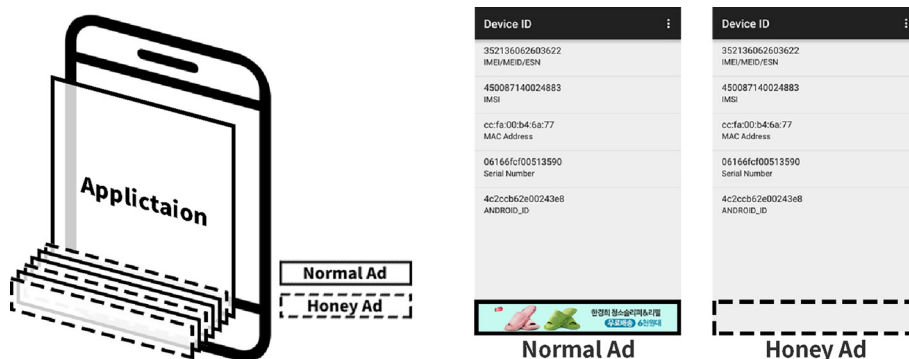

Fig. 6 Our proof-of-concept implementation for honey advertisement

Cho *et al. EURASIP Journal on Information Security* (2016) 2016:2

Page 9 of 9

Perhaps this is another example of "negative externality" [21] in the field of information security, which is an economic activity that imposes a negative effect on an unrelated third party.

8 Conclusions

This paper evaluates the potential risk of automated online fraud attacks in mobile advertising. Our experimental results show that 75 % of those networks (`Millennial Media`, `AppLovin`, `AdFit`, `MdotM`, `RevMob`, and `Cauly Ads`) are vulnerable to click fraud attacks.

To mitigate such automated attacks, we suggest three possible defense mechanisms: (1) filtering out program-generated touch events; (2) identifying click fraud attacks with faked advertisement banners; and (3) detecting anomalous behaviors generated by click fraud attacks. We also discuss why few companies were only willing to deploy such defense mechanisms with the economic aspects of security failure in the mobile advertising industry.

However, our current results are not enough to generalize our observations because of the limited number of tested ad networks. As an extension to this paper, we need to consider performing our tests on a large sample of ad networks.

In this paper, we only considered the simplest attack pattern where click events were successively generated within a fixed time interval and finally failed to successfully attack two ad networks. Therefore, as another extension to this work, we also plan to test other attack sequences patterns against secure ad networks.

Endnotes

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

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

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

Author details
[1]Department of Computer Science and Engineering, Sungkyunkwan University, Seobu-ro 2066, 16419 Suwon, Republic of Korea. [2]Samsung Electronics, Samsung-ro 129, 16677 Suwon, Republic of Korea.

References
1. S Dhar, U Varshney, Challenges and business models for mobile location-based services and advertising. Commun. ACM. **54**(5), 121–128 (2011)
2. I Leontiadis, C Efstratiou, M Picone, C Mascolo, in *Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications*. Don't kill my ads!: balancing privacy in an ad-supported mobile application market (ACM, 2012)
3. N Immorlica, K Jain, M Mahdian, K Talwar, in *Proceedings of The Workshop on Internet and Network Economics*. Click fraud resistant methods for learning click-through rates (Springer Berlin Heidelberg, 2005)
4. N Daswani, C Mysen, V Rao, S Weis, K Gharachorloo, S Ghosemajumder, Online advertising fraud. Crimeware Underst. New Attacks Defenses. **40**(2), 1–28 (2008)
5. B Stone-Gross, R Stevens, A Zarras, R Kemmerer, C Kruegel, G Vigna, in *Proceedings of the 2011 Conference on Internet Measurement Conference*. Understanding fraudulent activities in online ad exchanges (ACM, 2011)
6. J Crussell, R Stevens, H Chen, in *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*. MAdFraud: investigating ad fraud in android applications (ACM, 2014)
7. KC Wilbur, Y Zhu, Click fraud. Mark. Sci. **28**(2), 293–308 (2009)
8. N Kshetri, The economics of click fraud. IEEE Secur. Priv. **8**(3), 45–53 (2010)
9. B Miller, P Pearce, C Grier, C Kreibich, V Paxson, in *Proceedings of Detection of Intrusions and Malware, and Vulnerability Assessment*. What's clicking what? Techniques and innovations of today's clickbots (Springer Berlin Heidelberg, 2011)
10. H Haddadi, Fighting online click-fraud using bluff ads. ACM SIGCOMM Comput. Commun. Rev. **40**(2), 21–25 (2010)
11. B Kitts, JY 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. Real World Data Min. Appl. **17**(1), 181–201 (2015)
12. A Metwally, D Agrawal, A El Abbadi, in *Proceedings of the 16th International Conference on World Wide Web*. Detectives: detecting coalition hit inflation attacks in advertising networks streams (ACM, 2007)
13. V Dave, S Guha, Y Zhang, Measuring and fingerprinting click-spam in ad networks. ACM SIGCOMM Comput. Commun. Rev. **42**(4), 175–186 (2012)
14. H Xu, D Liu, A Koehl, H Wang, A Stavrou, in *Proceedings of 19th European Syposium on Reseach in Computer Security*. Click fraud detection on the advertiser side (Springer International Publishing, 2014)
15. P Pearce, AP Felt, G Nunez, D Wagner, in *Proceedings of the 7th Symposium on Information, Computer and Communications Security*. AdDroid: privilege separation for applications and advertisers in android (ACM, 2012)
16. S Shekhar, M Dietz, DS Wallach, in *Proceedings of USENIX Security Symposium*. AdSplit: separating smartphone advertising from applications (USENIX, 2012)
17. G Cho, J Cho, Y Song, H Kim, in *Proceedings of the International Workshop on Cyber Crime*. An empirical study of click fraud in mobile advertising networks (IEEE, 2015)
18. AM Azab, P Ning, J Shah, Q Chen, R Bhutkar, G Ganesh, J Ma, W Shen, in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. Hypervision across worlds: real-time kernel protection from the ARM trustzone secure world (ACM, 2014)
19. L Spitzner, *Honeypots: Tracking Hackers*. (Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002)
20. R Oentaryo, E-P Lim, M Finegold, D Lo, F Zhu, C Phua, E-Y Cheu, G-E Yap, K Sim, MN Nguyen, et al., Detecting click fraud in online advertising: a data mining approach. J. Mach. Learn. Res. **15**(1), 99–140 (2014)
21. T Moore, R Clayton, R Anderson, The economics of online crime. J. Econ. Perspect. **23**(3), 3–20 (2009)