KADİR HAS UNIVERSITY
GRADUATE SCHOOL OF SCIENCE AND ENGINEERING
PROGRAM OF INDUSTRIAL ENGINEERING

DETECTION OF FRAUDULENT ACTIVITIES IN MOBILE DISPLAY ADVERTISING

SAFİYE ŞEYMA KAYA

MASTER'S THESIS

İSTANBUL, JULY, 2018

DETECTION OF FRADUALENT ACTIVITIES IN MOBILE DISPLAY ADVERTISING

SAFİYE ŞEYMA KAYA

MASTER'S THESIS

Submitted to the Graduate School of Science and Engineering of Kadir Has University in partial fulfillment of the requirements for the degree of Master's in the Program of Industrial Engineering

İSTANBUL, JULY, 2018

DECLARATION OF RESEARCH ETHICS /
METHODS OF DISSEMINATION

I, SAFİYE ŞEYMA KAYA, hereby declare that;

- this Master's Thesis is my own original work and that due references have been appropriately provided on all supporting literature and resources;
- this Master's Thesis contains no material that has been submitted or accepted for a degree or diploma in any other educational institution;
- I have followed "Kadir Has University Academic Ethics Principles" prepared in accordance with the "The Council of Higher Education's Ethical Conduct Principles"

In addition, I understand that any false claim in respect of this work will result in disciplinary action in accordance with University regulations.

Furthermore, both printed and electronic copies of my work will be kept in Kadir Has Information Center under the following condition as indicated below:

The full content of my thesis/project will be accessible from everywhere by all means.

SAFİYE ŞEYMA KAYA

26.07.2018

KADİR HAS UNIVERSITY

GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

ACCEPTANCE AND APPROVAL

This work entitled **DETECTION OF FRAUDULENT ACTIVITIES IN MOBILE DISPLAY ADVERTISING** prepared by **SAFİYE ŞEYMA KAYA** has been judged and accepted by our jury as **MASTER'S THESIS**.

APPROVED BY:

Asst. Prof. Dr., Burak Çavdaroğlu (Kadir Has University)

Asst. Prof. Dr., Esra Ağca Aktunç (Kadir Has University)

Assoc. Prof. Dr., S. Tankut Atan (Işık University)

I certify that the above signatures belong to the faculty members named above.

Assoc. Prof. Dr., Demet Akten

Dean of Graduate School of Science and Engineering

DATE OF APPROVAL: 26/07/2018

TABLE OF CONTENTS

DETECTION OF FRAUDULENT ACTIVITIES IN MOBILE DISPLAY ADVERTISING

ABSTRACT

Most of the marketing expenditures in mobile advertising are conducted through real-time bidding (RTB) marketplaces, in which ad spaces of the sellers (publishers) are auctioned for the impression of the buyers' (advertisers) mobile apps. One of the most popular cost models in RTB marketplaces is cost per install (CPI). In a CPI campaign, publishers place mobile ads of the highest bidders in their mobile apps and are paid by advertisers only if the advertised app is installed by a user. CPI cost model causes some publishers to conduct some infamous fraudulent activities, known as click spamming and click injection. A click spamming publisher executes clicks for lots of users who haven't made them. If one of these users hears about the advertised app organically (say, via TV commercial) and installs it, this installation will be attributed to the click spamming publisher. In click injection, the fraudulent publisher's spy app monitors the user's activities in the app market to detect when a mobile app is downloaded on her device, and triggers a click attributed to the fraudster right before the installation completes. In this study, we propose a novel multiple testing procedure which can identify click spamming and click injection activities using the data of click-to-install time (CTIT), the time difference between the click of a mobile app's ad and the first launch of the app after the installation. In a sample set of publishers, we show that our procedure has a false-positive error rate of at most 5%. Finally, we run an experiment with 15263 publishers. According to the results of the experiment, a total of 1474 fraudulent publishers are successfully detected.

Keywords: Mobile Advertising, Fraud Detection, Click Spamming, Click Injection, Multiple Testing

MOBİL GÖRÜNTÜLEME REKLAMCILIĞINDA YAPILAN SAHTEKARLIK AKTİVİTELERİNİN BELİRLENMESİ

ÖZET

Mobil reklamcılıkta pazarlama harcamalarının çoğu gerçek zamanlı ihaleler aracılığı ile gerçekleştirilmektedir. Gerçek zamanlı ihalelerde, satıcıların (reklam yayınlayıcılarının) reklam alanları, alıcıların (reklam verenlerin) mobil uygulamalarına ait reklamlarının kullanıcı tarafından görüntülenebilmesi için ihaleye çıkartılır. Gerçek zamanlı ihalelerde en çok kullanılan fiyatlandırma modeli indirme başına fiyatlandırmadır. Bu modelde en yüksek fiyatı veren reklam verenin uygulaması yayınlayıcının uygulamasında gösterilir. Yayınlayıcıya ödeme bir kullanıcının bu yayınlanan reklamı tıklayarak indirmesi şartıyla yapılır. Bu model bazı yayınlayıcıların tıklama bombardımanı (click spamming) ve tıklama enjeksiyonu (click injection) olarak bilinen sahtekâr aktivitelerde bulunmasına neden olur. Tıklama bombardımanında yayınlayıcı gerçekte kullanıcı tarafından yapılmamış birçok tıklama üretir. Bu durumdan habersiz olan kullanıcı, reklamı yapılan uygulamayı farklı bir reklam kanalı (televizyon reklamı gibi) aracılığı ile öğrenip indirirse tıklama bombardımanını yapan yayınlayıcı bu indirmeden haksız yollarla para kazanmış olur. Tıklama enjeksiyonunda ise sahtekâr bir uygulamayı indirmek üzere olan bir kullanıcıyı takip edip, indirmeyi tamamlamadan hemen önce o kullanıcı üzerinden reklama tıklama gönderebilir. Dolayısıyla da bu indirme işlemi üzerinden haksız kazanç elde etmiş olur. Bu çalışmada çoklu test etme yöntemini kullanarak bu sahtekârlıkların belirlenebileceği bir yöntem önerdik. Bu yöntemde indirme ve ilk kez uygulamanın açılması arasında geçen zaman üzerinden istatistiksel karar verme yöntemleri kullanılarak sahtekârlar tespit edilmeye çalışıldı. Kullanılan yöntemde yanlış pozitif hata oranının en kötü ihtimalle %5 olması sağlandı. Önerilen yöntem 15263 yayınlayıcının üzerinde test edildi ve 1474 tanesinin sahtekârlık yaptığı tespit edildi.

Anahtar Sözcükler: Mobil Reklamcılık, Sahtekârlık Tespiti, Tıklama Bombardımanı, Tıklama Enjeksiyonu, Çoklu Test Etme

ACKNOWLEDGEMENTS

There are many people who helped to make my years at the graduate school most valuable. First, I would like to thank Asst. Prof. Dr. Burak ÇAVDAROĞLU, my major professor and thesis advisor. Having the opportunity to work with him over the years was intellectually rewarding and fulfilling. He has always encouraged me and recommended me to trust myself. He has always shown me the right path to reach success in my research.

Many thanks to other professors in my department, who patiently answered my questions and problems. I would also like to thank to my graduate student colleagues who helped me all through the years that are full of class work and exams. My special thanks go to Elif and Sirun whose friendship I deeply value.

The last words of thanks go to my family. I thank my parents for their patience and encouragement.

To my family

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

The time spent on mobile devices increased drastically in recent years and a significant portion of this time is spent in mobile applications. The widespread usage of both smartphones and mobile applications (apps) has led to the rapid growth of *mobile advertising*. Mobile display advertisements are displayed as banner images that are shown on mobile devices. They can appear in either web browsers or applications. If a mobile ad is designed for only mobile applications, it is called in-app advertising. Most of the marketing expenditures in mobile advertising are conducted through real-time bidding (RTB) marketplaces, in which the main objective of the buyers (advertisers) is to acquire the most app installation from the audience at the lowest cost and the main goal of the sellers (publishers) is to sell their ad spaces at the highest price. An advertiser decides the bid in the RTB marketplace and the attributes of the targeted users (such as the geolocation and demographics of the target users). Besides, ad agencies can be utilized for dissemination of these ads. An ad agency is a company that runs an ad campaign of a specific product with a predefined budget and campaign duration on behalf of clients. A publisher, which is usually the owner of a mobile app, on the other hand, sells the ad space of the app to advertiser at the winning price of an RTB auction. RTB marketplaces finalize an auction in milliseconds according to the bids of the advertisers. Thus, RTB allows advertisers and publishers to buy and sell ad space through real-time auction.

There are two main platforms in which RTB marketplaces are operated: ad exchanges and ad networks. An ad network, the RTB platform we focus on in this study, allows advertisers to publish their mobile advertising campaigns with a predetermined budget, campaign duration and a desired bid rate. Meanwhile, ad networks collect inventory of ad space from a range of publishers and sell it to advertisers with the highest bid offers in its RTB marketplace. Also, it allows advertisers to target desired customers. Ad networks usually offer different pricing models to the advertisers, such as cost per action, cost per install, cost per click, or cost per impression. In cost per impression model, advertiser is

charged when the advertisement appears on the screen. In cost per click model, bid price is paid, if the user just clicks advertising that is shown in an application by user. In cost per action model, a predetermined action by used will be required to charge the advertiser. In the cost per install model (CPI), the bid price is paid after installing the application that is shown as a display ad. The most popular pricing model among advertisers is CPI (Nieborg, 2016).

Even though mobile advertising is a billion-dollar industry, millions of dollars are lost because it is subjected to fraudulent activities. Mobile ad fraud is an attempt by fraudulent publishers to defraud advertisers for gaining undeserved profit. It is a popular subject in mobile marketing industry. In this study, we use statistical analysis to detect mobile frauds to prevent financial losses in mobile display advertising campaigns that prefer CPI pricing models.

1.1 Working Mechanism of Mobile Advertising

RTB landscape can be separated into two sides naturally; publisher side (supply side) and advertiser side (demand side). Demand side platform (DSP) is a software that is utilized to buy advertising in an automatic fashion by advertisers and ad agencies. In the advertiser side, DSP is requested by advertiser to run and manage an ad campaign. In the publisher side, process is started with a user interaction on publisher application. User preferences, context, location, and the mobile device information are sent by the mobile app to supply side platform (SSP) which is a software to sell mobile ads in an automatic fashion on behalf of publishers. Firstly, previous contracts are checked to send a request to an available contract of an advertiser. If there is not a contract or the advertisers who have contract are not interested in the impression, ad request is sent to ad exchange or ad network for RTB. Then, bid request is generated by RTB exchange or SSP for incoming ad requests and is sent to all subscribed DSPs. A bid request includes a couple of information such as a unique id of the request which is provided by RTB, the time of bid initiated, the current geographic location of the device, etc. The bid price is decided by DSP according to the ad campaign of the ad agency. The bid prices and bid responses are sent by all DSPs. A bid response contains information about id, price, currency, etc. After

the auction ends, RTB decides the winner of the auction based on bid prices. The winning note is sent to the winning DSP with the winning price which is the second highest offer (second price auction). The ad is requested from ad agency and sent to RTB. Finally, ad is forwarded to the publisher and user can see the ad on mobile application. The interactions among the involved parties can also be seen in Figure 1.1.

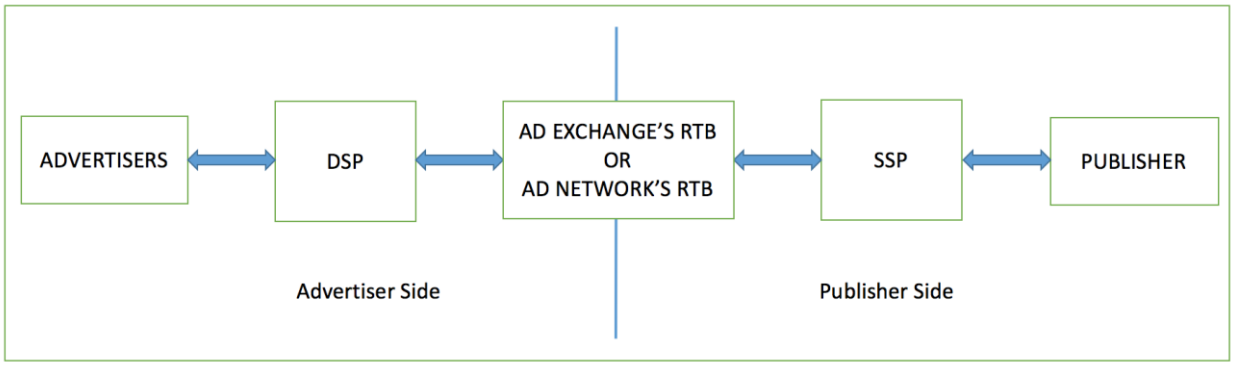

Figure 1.1. Working mechanism of mobile advertising

After that point, if the user clicks the ad and installs the advertised application, the advertiser has to pay the bid price according to the cost per install pricing model. In a CPI campaign, publishers place mobile ads of the highest bidders in their mobile apps in an effort to attract installation of the advertised application. The advertiser is charged the winning bid rate only when the advertiser's application is installed by the user.

After installation is verified by both the ad network and the advertiser, the ad network receives a portion of the CPI price for finding the publisher that grants the installation, and the publisher receives the rest. Verification of an installation takes place with the help of a system known as mobile ad attribution. Attribution is used to track the details of each mobile ad transaction such as the time stamp of the ad click and the ad installation. Attribution is also used to keep track of the publisher whom a succeeding installation should be attributed to. Each click by a user is associated with the publisher's app in which the click occurs. If there exist multiple clicks by a user before the installation of the advertised app, the latest attributed publisher will be paid by the advertiser due to the regulations of CPI pricing model.

1.2 Problem Definition

All these transaction and tracking activities are accomplished almost instantaneously in RTB marketplaces and they generate billions of dollars of revenue annually for mobile advertising industry. On the other hand, millions of dollars are lost because of the fraudulent activities of publishers receiving undeserved gain (Shields, 2016). Three main types of fraud in mobile advertising are (i) fake installations with bots and emulators, (ii) click injection, and (iii) click spamming. In the first type, using bots, fraudster tries to make fake installations more and more similar to genuine installations. In the second fraud type, fraudster's spy app detects when other apps are downloaded on a device and trigger clicks right before the installation completes, which inequitably attributes the installation to the fraudster's app. In the last type of fraud, fraudster executes clicks for lots of users who haven't made them. If a user installs the advertised app organically after hearing about the app in another advertising channel (for instance, a TV commercial), this installation will still be attributed to the fraudulent publisher according to the working mechanism of attribution system. Click spamming is the most common fraud type in app marketing and its most apparent harm is lost campaign budget of advertisers, by paying to the click spammer publishers for users who have never generated impressions in the publisher's app (Monasterio, 2017). Since click spamming captures organic traffic and then claims the credit for these users, it is also known as organic pouching.

Conversion Rate (CR) is a standard metric in app marketing that is calculated by dividing the total number of installations by the total number of clicks. A common characteristic of click spamming publishers is that their conversion rate is much lower than usual since spamming generates lots of false clicks which never end up with an installation. Although low CR is a good indicator for click spamming, a meaningful conversion rate emerges only after a significant campaign duration is completed and hundreds of thousands of clicks are attributed to the publisher. The lack of an early warning mechanism for click spamming causes the loss of considerable amount of campaign budgets for advertisers.

Another well-known way of identifying click spamming is to analyze the distribution of click-to-install times (CTIT) (Monasterio, 2017). Click-to-install time refers to the time difference between the click of a mobile app's ad by a user and the first launch of the app on the mobile device of the same user after the installation. It is easy to note that the spamming publisher can trigger false clicks but cannot trigger an installation. This makes click and installation events independent from each other and causes the click-to-install times of installations coming from a click spamming publisher to be distributed uniformly over time. Because click injection is more a sophisticated version of click spamming, CTIT can also be a good indicator for injection. In click injection, the fraudster can monitor a user by the help of a spy app and capture the moment when the user starts to install application. Therefore, the fraudster can trigger a click just seconds before the installation is completed. This action causes the distribution of CTIT values to take a uniform shape like in the case of click spamming. Even though CTIT is known to be utilized intuitively by many advertisers for filtering out spamming and/or injecting publishers, to best of our knowledge, a prescribed set of rules for fraud detection with CTIT has not been defined in the literature so far. In this study, we aim to derive a statistical method for the detection of click spamming and/or injecting publishers in real time by analyzing their CTIT data building up over time.

The rest of the paper is organized as follows: In Section 2, we provide a brief literature review on the fraud detection methods in web and mobile advertising and the statistical methods related to our analysis. In Section 3, the method of multiple testing is explained. In Section 4, we discuss the details of how our customized multiple testing approach can be used for the detection of click spamming and/or click injection activities. In Section 5, we present the data set and the experimental results. In section 6, we conclude by summarizing our findings and discussing future research opportunities.

2. PREVIOUS WORK

Fraudulent activities in digital advertisement is a research topic that has been widely investigated in the marketing and computer science literature. Soubusta (2008) provides information of analysis on click spamming in online advertising. He explained what click fraud is and how it works. Also, several solutions are offered for click spamming in online advertising. For example, according to Soubusta (2008), price models such as pay per action or pay per percentages of impressions can decrease the loss based on click spamming. Also, this study provides clear understanding on the effects of click spamming on the web. Classification based approach was developed by Daswani et al. (2008) to explain online advertising and online frauds similarly Soubusta. They classify not only the revenue types but type of spam activities in online advertising. They explain syndication and referral deals, besides the well-known revenue models (cost per mille, cost per impression, and cost per action) for online advertising activities. According to Daswani et al. (2008) there are three main type of spams which are impression spam, click spam, and conversion spam. Furter, attack types, some countermeasures, and economics of click fraud are discussed in the same study.

There are different types of fraud. For example, impression fraud is basically caused by pay per view pricing model on the web. Springborn and Barford (2013) describe the characteristic of pay per view ecosystem and developed a method to distinguish fraudulent impressions from non-fraud ones. For developing this method, they made analysis of purchased traffic on websites and collected data from these websites. One of the most prevalent fraud type, which both web and mobile platforms suffer from, is click spamming. A lot of studies are conduced to understand the clicking behaviors. Hill et al. (2014) develops tools and techniques to detect invalid clicks in websites. They provide a system to obtain historical click quality characterization based on web analytic data. In this way, the system identifies click abnormities. Also, Perera et al. (2013) argues that

click patterns can be utilized to identify fraudulent activities. They provided an approach to detect fraud by using a set of features which are derived from existing attributes and used learning algorithms to understand differences of click patterns between fraud and legit publishers. The mobile advertising data are complex and include heterogeneous information, and complicated patterns with missing values. Therefore, Fraud Detection in Mobile Advertising (FDMA) 2012 Competition was organized. 127 teams joined the competition from more than 15 countries. Oentaryo et al. (2014) provided information about competition that include data set, task objectives and evaluation of results of competitors ranked in the first three places. According to competition results, data mining based fraud detection can be usable in practice. Immorlica et al. (2005) used machine learning techniques that are based on click through rates to detect click fraud in pay-per-click pricing model. Fraudsters usually conduct click spamming on the web by disseminating malicious software (malware) that are capable of generating fake click on behalf of the infected users. Blizard and Livic (2012) outlined an example analysis of a click-spamming malware and showed that the malware can cause a loss on the order of hundreds of thousands of dollars for a 3-week period. Jain and Talwar (2007) argued that dual pricing can reduce the effects of fraudulent activities in real time auctions. Iqbal et al. (2018) presented a method for fighting click-fraud by detecting botnets with automated clickers from the user side. They also evaluated the performance of their proposed method by integrating it into desktop operating systems. Zingirian and Benini (2018) showed a vulnerability of the pay-per-click model in web advertising and proposed a statistical tradeoff-based approach to manage this vulnerability. There are a lot of patents to find and/or identify click fraud on the web. One of them is provided by Kitts et al. (2008). They developed methods and systems to detect automated click fraud programs. When a request is received for a web page, the probability of being a genuine bot user is determined. A score is determined according to historic behavior of the related user. In this way, user who is human can be separated from the user who is bot. Another patent to prevent click fraud in online adverting is taken out by Linden and Teeter (2012). They provided a method that includes server side and client side codes to achieve their goal which is identification of valid and invalid clicks. Smith et al. (2011) developed systems and methods for detecting click spam in web advertising and patented this methodology.

Their system identifies normal users visiting a web site and determines an occurrence of spamming on the web site based on the identified normal users.

Pay-per-click pricing model of web advertising requires instant payment to the publisher upon click. Click spamming has a direct negative effect on the profitability of the advertisers on web advertising whereas in the CPI pricing model of mobile advertising, click spamming can affect an advertiser only if click ends up with an installation. Therefore, fraud in mobile advertising is a relatively new research area when compared with fraud in web advertising.

Mobile application markets have many freely distributed applications that are supported by in-app advertisements. Most of the fraudulent activities are performed by the publishers of these applications. Both placement fraud and bot fraud in these apps cause impression and unintentional clicks from users (Liu et al. 2014). Liu et al. (2014) investigated display fraud by analyzing UI of apps to detect unintentional clicks for increasing ad revenue. However, this technique cannot determine clicks that are triggered in the background. Unintended click can be performed both in foreground and background (Crussell et al. 2014).

Cho et al. (2015) made an automated click generation attack on eight popular ad networks and showed six of them vulnerable to this type of attacks. Cho et al. (2016) expanded their previous study. They suggested defense mechanisms and discussed economic aspect of security failure. Dave et al. (2012) conducted large-scale measurement study on major ad networks about click spamming and proposed a methodology to measure click spamming rate for advertisers. Badhe (2016) suggested a new system which consists of a server side solution for click fraud. Badhe (2016) offered an exchange mechanism that scans the ads before passing them over to the end mobile device. This mechanism provides checking for any auto redirection to different domain from initial domain where all ad assets requested. However, ad exchanges have to deal with billions of ads daily. Checking these ads one by one could be infeasible. This problem can be solved with taking random sampling according to Badhe (2016). Gupta et al. (2014) discloses different types of mobile frauds. They argued the source of requests may be used for

distinguishing valid and invalid requests in order to detect frauds. Monasterio (2017) proposed a histogram for the click-to-install time distribution of non-fraudulent publishers and utilizes a fitting test to tag click spamming activities. However, this fitting test method can only be utilized at the end of ad campaign duration and cannot be used in real time.

3. MULTIPLE TESTING

In this chapter, the statistical methods to detect frauds is examined and suitability of multiple testing in our case is discussed. Let us first introduce the histogram of click-to-install time (CTIT) values for two publishers as an example of how the distribution of CTIT values may differ in legit and fraudulent publishers. In Figure 3.1, the histogram on the left shows the distribution of click-to-install times for a legit publisher. It can be noted that most of the installations are accomplished within the first hour after the click event occurs since they decide to install and launch the app shortly after they deliberately click the publisher's ad. The histogram on the right, on the other hand, demonstrates the distribution of click-to-install times for a fraudulent publisher. This publisher spams lots of users with lots of clicks, and a few users unaware of this click event (and, thus, the advertised app) will occasionally install the app after hearing about it from other marketing channels or via word of mouth. Hence, the time between click and installation events can be weeks, or even months, which results in a hump on the right of the graph of fraudulent publisher.

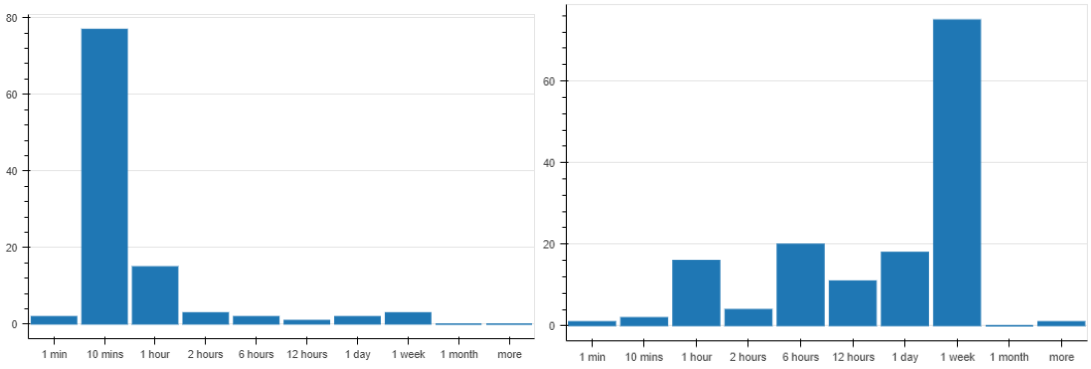

Figure 3.1. The distribution of CTIT values for a legit publisher (left) and a fraudulent publisher (right)

Publishers dealing with click injection can also be distinguished from legit publishers via distribution of CTIT values like in the case of click spamming. In Figure 3.2, the histogram belongs to a publisher who engages in click injection. There is an abnormal hump at the left of the graph (second bar in the histogram) due to the nature of the click

injection. A click injection fraudster can trigger a click after the installation. Therefore, the time between click and first launch is usually inclined to be less than 20 seconds. This is the reason of CTIT values accumulated on the left side of the histogram.

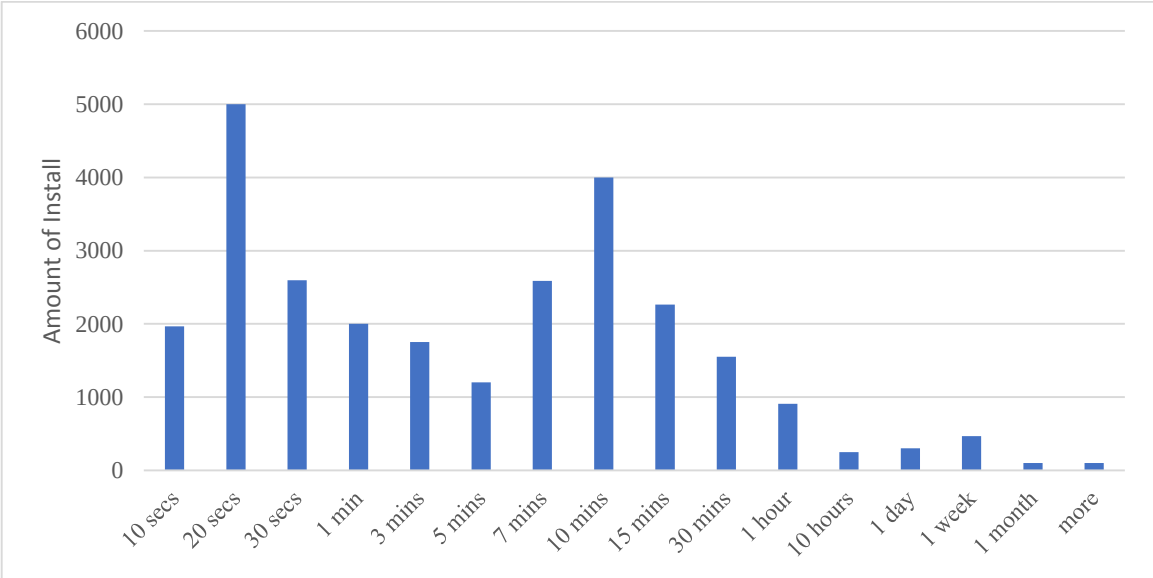

Figure 3.2. The distribution of CTIT values for a fraudulent publisher (click injection)

Another interesting fact about the distribution of click-to-install times lies behind the descriptive statistics of these values. Table 3.1 provides the sample size, mean, standard deviation, median, and range for the CTIT values of the same two publishers discussed earlier. The table also shows the statistics for the all-time installations of a DSP company for benchmark purposes. The mean of CTIT vales for a legit publisher can still be very high (approximately 4.75 hours) due to a few installations with very high CTIT values. This situation is not an indicator of click spamming since some users forget to launch a mobile application after the installation and thus even non-fraudulent publishers may rarely run across very high CTIT values. Therefore, “mean” is not a reliable statistic to make any deduction about spamming. The median value of fraudulent publisher, on the other hand, is quite large compared to the medians of DSP and legit publisher according to the table. Indeed, a very large median (approximately 27 hours in this instance) means that at least half of the sample has an unacceptable level of CTIT. A large median value also explains why the histogram of the fraudulent publisher is skewed to the right in Figure 3.1. Further, the median value of the fraudulent publisher who makes click injection is quite small. These results suggest that a statistical test measuring the positive

deviations from a sufficiently large or small "median" can be confidently used to detect fraud publishers.

The non-parametric sign test for a median (Sprent, 1989) with H_0: $\eta = \eta_0$ and H_a: $\eta > \eta_0$ (or H_a: $\eta < \eta_0$) is an effective way of deciding whether the median of CTIT values for a publisher (η) takes on a particular value (η_0) or a value greater (or less) than η_0. The main question here is how to decide η_0. We have to select a sufficiently large value for η_0 such that we can safely accuse the publisher of spamming if the null hypothesis is rejected. Similarly, sufficiently small value has to be selected to accuse publisher of making click injection.

Table 3.1. The descriptive statistics of CTIT values for legit and fraudulent publishers

	All-time Installs of a DSP	Legit Publisher	Fraudulent Publisher (Spamming)	Fraudulent Publisher (Injection)
Sample Size	1602268	106	148	622
Mean (sec)	68991	17016	187398	14
Median (sec)	250	236	96941	2
Standard Deviation (sec)	553750	78452	338678	18
Minimum (sec)	1	35	51	1
Maximum (sec)	15851602	523212	3730800	86
Range (sec)	15851601	523177	3730749	85

The following sign tests are designed to detect two main types of mobile fraud. While the test (1) is for click spamming, the second test is for click injection.

$$H_0: \eta = \eta_0 \text{ seconds} \quad H_a: \eta > \eta_0 \text{ seconds} \qquad (1)$$

$$H_0: \eta = \eta_0 \text{ seconds} \quad H_a: \eta < \eta_0 \text{ seconds} \qquad (2)$$

However, our analysis shows that, when the test is applied for the all-time installs of a publisher (all the installations attributed to a publisher during the lifetime of an ad campaign), there is a significant chance of the publisher passing the test even though it is fraudulent (i.e. probability of type-II error is large). This situation occurs mainly because some publishers mix both click spamming and legit activities together in order to disguise

their fraud. Therefore, it is possible for a fraudulent publisher to have remarkable number of very large CTIT values even though the median CTIT value is still less than η_0 seconds (or greater than η_0 seconds in click injection case). Besides, conducting the test for the all-time installations of a publisher would mean to evaluate the publisher after an ad campaign has ended. Filtering out a spamming publisher from future campaigns is still beneficial in the long run, but it cannot prevent the advertiser from paying for the installations that are already attributed to the fraudster at the recent campaign.

One way to overcome these challenges is multiple testing, the testing of more than one independent hypothesis. In multiple testing, instead of running a single sign test for a publisher at the end of campaign duration, we periodically run sign tests as new installations arrive from the publisher. This enables us to detect even occasional click spamming activities of fraudulent publishers in earlier stages of the campaign.

However, if one plans to make a decision by applying multiple testing, s/he should be extra cautious about false-positive decision making. Assuming that the type-I error of a single sign test is α, the probability of not making type-I error is $(1-\alpha)$ because they are complementary events. For instance, if significance level $\alpha = 0.10$, then not making type-I error is $(1-\alpha) = (1-0.10000) = 0.90000$. Let say we have two independent tests with $\alpha = 0.10000$. Probability of not making type-I error for both of them is $0.90000 * 0.90000 = 0.81000$. For three independent tests, the probability will be $0.90000 * 0.90000 * 0.90000 = 0.72900$. The probability of not making type-I error for m independent tests is calculated with $(1-\alpha)^m$. Therefore, the probability of at least one false positive error among m independent sign tests (family-wise error rate) is $\bar{\alpha} = 1-(1-\alpha)^m$. This means that the probability of having at least one rejected null hypothesis converges to 1 as the number of tests increases. In other words, if we assume one rejected null hypothesis in multiple testing is adequate to accuse a publisher of click spamming, we will falsely blame the publisher for fraud even though it is most probably legit.

In the literature, there are classical multiple testing methods such as Šidák correction (Šidák, 1967), Holm method (Holm, 1979), and Bonferroni correction (Bland and Altman, 1995), which prevent large probability of rejecting some of the true null

hypotheses. Also, we developed a new procedure to make sure limited family-wise error rate at acceptable level.

3.1 Šidák Correction Method

The method is utilized to avoid problem of multiple comparisons. It is one of the simple methods to keep family-wise error rate for independent tests under control. The m^{th} null hypothesis is rejected, when p-value is less than $\bar{\alpha} = 1 - (1 - \alpha)^{1/m}$ for each test. If all null hypotheses are true, type-I error will be exactly α. For example, let's say we have 3 independent null hypotheses and $\alpha = 0.05000$. Family-wise error is calculated as 0.05000, 0.02532 and 0.01695 respectively. If the p-values of matching tests are greater than the corresponding $\bar{\alpha}$, type-I error is equal to 0.05000. In addition, confidence interval is calculated as $100(1 - \alpha)^{1/m}$ for matching test decision.

3.2 Bonferroni Correction Method

This method is used for exactly the same reason as in Šidák method. Holm developed the method originally in 1979. In Holm method, significance level is divided by both total number of the independent tests and the index k, which is used if the first p-value is not low enough to validate rejection at $\bar{\alpha} = \frac{\alpha}{m+1-k}$. This index is accepted as 1 in Bonferroni Correction. Therefore, the family-wise error controls with $\bar{\alpha} = \frac{\alpha}{m}$. This means that if the p-value of m^{th} test is less than $\frac{\alpha}{m}$, $H_0^{(m)}$ that represents the null hypothesis of mth test is rejected, otherwise $H_0^{(m)}$ fails to be rejected. Table 3.2 presents the calculated family-wise error rates by Šidák and Bonferroni methods when α is assumed to be 5%. The first column shows the number of tests. The family-wise error rates of Šidák method according to varying total number of tests is shown in the second column, while the rates of Bonferroni method are demonstrated in the third column. As seen in the table, the family-wise error for a single test is the same for both correction methods. However, family-wise error rate that is calculated with Bonferroni method is always less than the family-wise

error rate of Šidák method after the first test. Šidák method is slightly less stringent than Bonferroni correction.

Table 3.2. Family-wise error rates for Šidák and Bonferroni methods

Number of Test	Šidák	Bonferroni
1	0,05000	0,05000
2	0,02532	0,02500
3	0,01695	0,01667
4	0,01274	0,01250
5	0,01021	0,01000
6	0,00851	0,00833

3.3 Method of Successive Runs

In this study, we propose a new multiple testing procedure that has an improved ability to detect click spamming fraud compared to these classical methods, while still keeping the confidence of the procedure (i.e. probability of not making false-positive decisions) sufficiently high.

In our multiple testing procedure, we run successive sign tests in real time while the campaign is still running. In other words, we run a sign test given in (1) for every n installations of a publisher to monitor the legitimacy of its installations. If we had decided the publisher is fraudulent by only one rejected null hypothesis in multiple testing, as mentioned earlier, we would have made a considerable number of false-positive decisions. Instead, in our procedure, we aim a family-wise error rate of $\bar{\alpha} = 0.05000$ in the worst-case.

In our study, the multiple testing procedure for detecting click spamming activity has been implemented in two steps as follows.

- STEP 1. We assume the significance level of $\alpha = 0.05000$ for each sign test for median, whose sample size is selected to be $n = 10$. Namely, we run the

hypothesis test of (1) for every 10 incoming installations and conduct $m < \lceil N/10 \rceil$ tests in total, where N is the total number of installations attributed to the publisher during the campaign duration.

- STEP 2. We define a rule which identifies the spamming publishers due to the result of m sign tests with a family-wise error rate of $\bar{\alpha} = 0.05000$. According to the rule, a publisher is determined to be fraudulent if it fails r successive sign tests, each with a significance level of α, among m tests. We do not have to run these tests till the end of the campaign duration (i.e. m does not have to be equal to $\lceil N/10 \rceil$) since we may run across r successive rejected hypotheses in earlier stages of the campaign. Note that the value of r has to be updated as the number of tests (m) increases with the incoming installations. If we set a constant value for r, the family-wise error rate would continuously increase and eventually be much higher than our target value, $\bar{\alpha} = 0.05000$, as the number of tests, m, increases.

In order to compute the value of r for varying levels of m, we need to utilize the theory of success runs introduced by Feller (1968). Let r be a positive integer and let ε denote the occurrence of a success run of length r in m Bernoulli trials, each with a success probability of α. According to Feller (1968), the probability of no success run of length r in m trials (denoted by q) can be approximated by Eq. (3)

$$q \approx \frac{1 - \alpha x}{(r + 1 - rx)(1 - \alpha)} \cdot \frac{1}{x^{m+1}} \tag{3}$$

where x is the positive root of Eq. (4), which is not equal to $1 - \alpha$.

$$1 - x + (1 - \alpha)\alpha^r x^{r+1} = 0 \tag{4}$$

Hence, the probability of at least one success run of length r in m trials is given by $p = 1 - q$.

For example, assume we conclude a publisher's median CTIT is greater than 2 hours (i.e. the publisher is fraudulent) if the null hypotheses of $r = 3$ successive sign tests are rejected among $m = 300$ tests. When we let $\alpha = 0.05$ and $r = 3$ in Eq. (4), x can be found as 1.000119. Replacing this value of x in Eq (3), the probability of falsely concluding the publisher to be fraudulent (family-wise error rate) can be calculated as $\bar{\alpha} = p = 0.0348$.

In the second step of our procedure, we determine the rule of how many successive rejected hypotheses are enough for concluding a publisher is a spammer for different numbers of tests. The main objective of this rule is to guarantee that the probability of falsely accusing the publisher does not exceed a family-wise error rate of $\bar{\alpha} = 0.05000$. For example, for $r = 3$, we look for the value of "m" which makes p close to $\bar{\alpha} = 0.05000$ as much as possible. For varying values of m in Eq (3), the value of $p = q - 1$ is given in Table 3.3. As it can be noted in the table, p can get the value which is closest to $\bar{\alpha} = 0.05000$, when $m = 434$.

Table 3.3 The family-wise error rates respectively the values of m

m	p
425	0,04902
426	0,04913
427	0,04925
428	0,04936
429	0,04947
430	0,04959

m	p
431	0,04970
432	0,04981
433	0,04992
434	0,05004
435	0,05015
436	0,05026

Table 3.4 tabulates how the rule is being applied. $\bar{m}$ represents the number of tests required to have a successive rejected hypothesis of length r with a probability of $\bar{\alpha} = 0.05$. $[m^l, m^u]$ denotes the interval for the total number of tests where successively rejected hypotheses of length r are sought for click spamming. According to Table 3.4, our multiple testing procedure works as follows. When the first 10th installation attributed to a publisher is registered, we run the sign test given in (1) for the CTIT values of this sample. If the null hypothesis is rejected, we conclude that the publisher is a click spammer (since $m = 1$ when $r = 1$). Otherwise, we wait until another 10 installations are registered. When the next 10 installations are accumulated, we run the sign test again for the second set of 10 installations. Starting from the 2nd sign test to the 22nd test, we search for 2 successively rejected hypotheses. If we observe any "two rejected hypotheses in a row" until the 22nd test, we conclude that the publisher is a click spammer. Otherwise, we continue to monitor the installations attributed to the publisher. Likewise, we search for 3 successively rejected hypotheses from the 23rd test to the 434rd test, and 4 successively rejected hypotheses from the 435th test to the 8524th test. We have never needed to go beyond the 8524th test, since none of the publishers in our experiment has

more than $8524 * 10$ attributed installations. If we conclude that a publisher is fraudulent at any point according to this procedure, the decision is guaranteed to have a false-positive error that is at most equal to the family-wise error rate of $\bar{\alpha} = 0.05000$. If we do not observe any "r rejected hypotheses in a row" during the course of our procedure, we can conclude that there has not been enough evidence that the publisher is a click spammer.

Table 3.4. The rule of multiple testing procedure

r	$\bar{m}$	$[m^l, m^u]$
1	1	[1,1]
2	22	[2,22]
3	434	[23,434]
4	8524	[435,8524]

3.4 Accuracy Test for Method of Successive Runs

We will now discuss the results of our accuracy tests conducted with a data set of app installations attributed to several publishers. This data set was supplied by a DSP company which provides a self-service mobile advertisement platform to its customers for managing their own ad campaigns. In our experiment, we run the tests with 30 publishers from 30 different ad campaigns. Half of the publishers have been blacklisted as click spamming publishers by the DSP company due to prior experience. The remaining 15 publishers are known to be non-fraudulent that always bring in legitimate installations.

Table 3.5 summarizes the data used and shows the results of our test. Each publisher is represented with a row in the table. The grayed-out rows denote the publishers that are in fact fraudulent. The first five columns provide the publisher ID, the number of total installations during the campaign, the campaign duration, the mean value of all CTITs during the campaign, and the median value of all CTITs, respectively. The sixth column shows the total number of rejected tests during the campaign when we were applying method of successive runs. The number of rejected tests does not have to be zero for a non-fraudulent publisher. For example, first publisher is not identified as fraudulent,

although it has a total of 2 rejected tests. The reason it is not marked as a click spammer is that the rejected tests are not successive. The last column which has three parts specifies the order of the first test indicating a click spamming activity. The first part shows our method, the second one represents Bonferroni, and the last one is for Šidák. For instance, the second publisher is accused of click spamming for the first time in its $657 * 10 = 6570^{th}$ attributed installation with method of successive runs, since r= 4 consecutively rejected tests are observed at the 657^{th} test in our multiple testing procedure. A dash sign ("−") in the sixth column indicates that any evidence of click spamming has not been found during the campaign.

According to the results of the experiment, our multiple testing procedure does not make any false-positive error. In other words, none of 15 legit publishers is accused of click spamming. Since we design the procedure in such a way that it does not have a false-positive error greater than the family-wise error rate of $\bar{\alpha} = 0.05000$, this is an expected outcome. On the other hand, in three tests, we observed false-positive errors which were made by Šidák and Bonferroni Corrections. As seen in the Table 3.5, even though Publisher 1, 9, 19 and 28 are legit publishers in reality, they are marked as fraud by Šidák and Bonferroni Correction methods. For example, the first publisher is found as a spammer in the 29^{th} test by both of them because all ten installations for the test came after the first two-hour period and the p-value of the 29^{th} test is calculated as 0.00098. The family-wise error which belongs to this test is $\bar{\alpha} = \frac{0.05}{29} = 0.00172$ for Bonferroni method, $\bar{\alpha} = 1 - (1 - 0.05)^{\frac{1}{29}} = 0.00177$. Therefore, $H_0^{(29)}$, which is the null hypothesis of 29^{th} test is rejected and the publisher looks like a spammer. Even though, $H_0^{(29)}$ was rejected, the publisher was found legit in our method because of our rejection rule. As seen in Table 3.5, there are 2 reject decisions in method of successive runs, 3 successive rejects are needed to accuse someone of being a fraud after the 22^{nd} test. Hence, since at least one of the 28^{th} and 30^{th}, or 30^{th} and 31^{st} tests were not rejected, our procedure decided the publisher is legit correctly.

Table 3.5. Accuracy Test Results

						The Order of the Test Indicating Click Spamming		
Publisher	Number of Total Installs	Campaign Duration (days)	Mean CTIT (seconds)	Median CTIT (seconds)	Total Number of Rejected Tests In Method of Successive Runs	Method of Successive Runs	Bonferroni Correction Method	Šidák Correction Method
1	308	36	138454	64	2	-	29	29
2	8424	79	4433	201	172	657	-	-
3	749	3	6983	727	0	-	-	-
4	1038	11	25180	1103	0	-	-	-
5	94	2	1441	1462	0	-	-	-
6	2420	23	2808	1869	0	-	-	-
7	1351	175	14223	2376	4	135	-	-
8	288	163	15881	2607	2	-	-	-
9	875	44	4068	3011	0	-	21	21
10	1876	174	11426	3590	3	-	-	-
11	265	6	3332	3666	2	14	14	14
12	1091	128	10995	4293	6	67	-	-
13	249	7	9993	4487	0	-	-	-
14	373	4	5204	5018	0	-	-	-
15	133	39	6665	5383	0	-	-	-
16	827	34	15788	5919	24	6	5	5
17	416	29	8139	6240	1	-	-	-
18	1075	3	13493	6406	42	17	16	16
19	361	58	11967	6999	5	-	6	6
20	295	17	95731	7173	8	13	12	12
21	400	24	64212	7330	5	40	7	7
22	1045	79	29884	7879	31	24	9	9
23	335	118	19482	8235	0	-	-	-
24	150	9	17020	8999	1	-	-	-
25	62	5	22196	9441	1	-	3	3
26	2487	94	23964	9945	54	15	48	48
27	90	8	16608	10315	1	-	9	9
28	610	32	25412	10732	10	-	9	9
29	877	6	19497	11383	19	19	18	18
30	2621	44	31341	19686	97	15	3	3

Although there are 4 type-I errors that are found by Šidák and Bonferroni Correction methods, publisher 25 is detected by these methods unlike the method of successive runs. Since Šidák and Bonferroni Correction methods slightly stringent than method of successive runs, they detect most of the fraudulent publishers earlier than our method. However, they have much higher type-I error rates. For example, while publisher 21 is detected with the last installation by our method, Šidák and Bonferroni Correction methods detected it earlier with the 7^{th} test. As seen in the table, the orders of all of the test indication click spammers are the same for Šidák and Bonferroni, because values of the family-wise error are close to each other. For instance, the family-wise error of the third test is 0.01695 for Šidák Correction, while it is 0.01667 for Bonferroni method.

As seen Table 3.6, the results also indicate that while 12 out of 15 fraudulent publishers have been successfully detected with method of successive runs, Šidák and Bonferroni correction methods have detected 11 out of 15 fraudulent publishers. On average, while we detect the click spamming activities before 32% of the entire set of installations have arrived with the method of successive runs, click spamming activities are detected before 6.70% of the total installations have arrived with the other methods. In the best case (Publisher 30), the click spamming is detected before $(15 * 10)/2621 = 6\%$ of the total installations have arrived. The best case of the other methods is observed in the case of the same publisher. The spammer is detected before $(3 * 10)/2621 = 1.14\%$ of the total installations have arrived. In the worst case for our method (Publisher 21), the click spamming can be detected only after $(40 * 10)/400 = 100\%$ of the total installations have arrived. Publisher 27 is detected after the last installation is arrived, being the worst case of Šidák and Bonferroni methods. This result suggests that we have recognized that Publisher 21 and 27 are fraudsters at the very end of the ad campaign. Even though the detection of these spamming publishers does not help us in this experimented campaign, filtering out the publisher from the future campaigns will protect us against its prospective fraud. Apart from these results, Publisher 2 is also worth mentioning because it is blacklisted as click spammer in spite of its relatively low median CTIT value. If we had conducted the sign test for once with the all-time installations of this publisher, we would have concluded it is not fraudulent due to the median CTIT of 201 seconds. This example shows why the multiple testing procedure is superior to single testing.

Table 3.6. Summary of accuracy test results

Method	Detected Fraud	False Positive Decision Rate	Average Detecting Installation
Šidák	11/15	27%	6.70%
Bonferroni	11/15	27%	6.70%
Our Method	12/15	0%	32%

On the other hand, we fail to detect the click spamming activities of the remaining four fraudulent publishers (Publisher 5, 24, 25, and 27) by our method. Publisher 2, 5, 7, 12 could not be detected by the other methods during their campaign period, which makes the false-negative error rate of the experiment to be $\bar{\beta} = 11/15 = 73\%$. The common characteristic of all four publishers 5, 24, 25, and 27, whose click spamming activities have not been detected, is the low number of installations due to the short campaign duration. We observe that most of the clicks attributed to these publishers had not ended up with installations at the time point we collected their data for our experiment. Note that the bid price is still paid to publishers, if the click event takes place during the campaign interval regardless of the time of the installation event. Most probably, these clicks will never yield a legit installation, but can still bring in undeserved money for these publishers if any user installs the advertised app organically via another advertising channel. On the other hand, common characteristic of the other publishers whose fraudulent activities are not detected is that their spam attacks started near the end of the campaign durations except Publisher 5. This situation is based on the nature of Šidák and Bonferroni Correction methods. In these methods family-wise errors decrease continuously. However, p-value of tests has a limit to decrease because of the sample size. Hence, if the sample size is 10 for tests, p-value can be 0.00098 in the worst case when all ten installations have arrived after at least 2-hours from the related click. After a certain amount of testing, family-wise error will be always smaller than the p-value. Therefore, null hypothesis will not be rejected after that point because of the reject rule which says that the null hypothesis is rejected, if the p-value is bigger than family-wise error for that test. While family-wise error for Bonferroni method is always smaller than the p-values of tests after the 52nd test whose p-value is 0.00096, limit for Šidák method is 53rd test with $\bar{\alpha} = 0.00097$. After 53rd test, p-value of test will consistently be greater

than family-wise error independently from CTITs. For example, family-wise error is calculated as $\bar{\alpha} = 0.00050$ with Bonferroni method for the 100^{th} test. The publisher cannot be accused of being fraud even if the CTITs of 10 installations are greater than 2-hours because p-value of 0.00098 is greater than $\bar{\alpha} = 0.00050$. Even though Šidák and Bonferroni methods are useful for short-term campaigns, they are useless for long term campaigns and for publishers who have approximately more than 530 installations in a campaign because of their stringency.

Unfortunately, for the short-term campaigns, our multiple testing procedure is not likely to detect the click spamming publishers since high-valued CTITs indicating spamming have not yet been generated at the time we collect the data. Nevertheless, these campaigns are short-dated and usually bring relatively small number of falsely-attributed installations from their publishers. Hence, we can safely assume the total loss due to these installations is still in an acceptable level.

4. DETECTION OF FRAUD USING MULTIPLE TESTING

In this section, steps of detecting click spamming and click injection are explained. Insufficiency of using single testing is showed in the previous section. Mean of CTITs is not a useful source to detect frauds. Even median of CTITs which gives a better understanding of the publishers' actions can mislead the advertisers when they decide fraudulent activities. For example, while having a median that is bigger than 2-hour can be proof of click spamming, having a small median does not guarantee that the publisher is not a spammer. Therefore, using non-parametric single sign test for median of CTITs can cause type-II errors often in the case of small median of CTITs. On the other hand, multiple testing is better than in terms of both type-I and type-II errors. Moreover, multiple testing methods provide an opportunity to test the publisher in real time unlike using fitting test for detection of fraud. The end of campaign duration has to be waited to make fitting test. This causes loss during the campaign durations. On the other hand, fraudsters can be detected during the campaign duration and they can be blocked to prevent financial losses with multiple testing methods through applicability in real time. The method of successive runs is better than other correction methods of multiple comparison because of the limitations of Šidák and Bonferroni methods. These methods do not have ability to detect frauds in long term campaign durations and for the publishers who have more than 530 installations in a campaign. Furthermore, these methods are vulnerable in terms of type-I errors unlike our method. Type-I error is a more dangerous error than type-II. That is because, while type-I error represents accusing a publisher who is legit of being fraud, type-II error means that a publisher who is a fraudster in reality is found legit. Even though a publisher is found legit falsely in a campaign (type-II error), it can be detected in the other campaigns. However, if a publisher is detected as a fraud, it is blocked permanently. Therefore, there is no compensation for making type-I error. This is the reason for selecting method of successive runs to detect fraud types in this chapter instead of other methods. Also, publishers should be evaluated campaign by campaign because legit publishers can decide to start fraudulent activities after a while.

Therefore, every installation should be examined for both click spamming and click injection.

4.1 Detection of Click Spamming

In this chapter, detection of click spamming is explained step by step. The method of successive runs is utilized to detect spammers. Pseudocode is shown in below:

- Start Campaign n
- Set $Install\ Counter\ =\ 0$
- Set $Minus = 0$
- For each arriving installation for n^{th} campaign
    - $Install\ Counter\ =\ Install\ Counter\ +\ 1$
    - $CTIT_{ikn} =\ Install\ Time_{ikn} - Click\ time_{ikn}$
    - For each test $t \in T$
        - If $Install\ Counter\ \%\ 10\ =\ 0$
            - $Sign\ =\ CTIT_{ikn}^{t} - 7200$
            - If $Sign\ <\ 0$
                - $Minus\ =\ Minus\ +\ 1$
        - Calculate $Pvalue$ of Test t
        - $Pvalue(t) = \sum_{k=0}^{Minus} \binom{sample\ size}{k} p^k (1-p)^{sample\ size-k}$
        - If $t = 1$
            - If $Pvalue\ (1) < \alpha$
                - Publisher k is a fraudster
                - Block Publisher k
            - Else
                - Wait for the next ten installs and conduct Test 2
        - If $t \geq 2$
            - $If\ Pvalue(t)\ <\ \alpha\ \&\ Pvalue(t+1)\ <\ \alpha$

            - Publisher k is a fraudster
            - Block Publisher k
        - Else
            - Wait for the next ten installs and conduct Test $t + 2$
    - If $t > 22$
        - If $Pvalue(t) < \alpha$ & $Pvalue(t+1) < \alpha$ & $Pvalue(t+2) < \alpha$
            - Publisher k is a fraudster
            - Block Publisher k
        - Else
            - Wait for the next ten installations and conduct Test $t + 3$
    - If $t > 434$
        - If $Pvalue(t) < \alpha$ & $Pvalue(t+1) < \alpha$ & $Pvalue(t+2) < \alpha$ & $Pvalue(t+3) < \alpha$
            - Publisher k is a fraudster
            - Block Publisher k
        - Else
            - Wait for the next ten installations and conduct Test $t + 4$
    - If $t > 8524$
        - If $Pvalue(t) < \alpha$ & $Pvalue(t+1) < \alpha$ & $Pvalue(t+2) < \alpha$ & $Pvalue(t+3) < \alpha$ & $Pvalue(t+4) < \alpha$
            - Publisher k is a fraudster
            - Block Publisher k
        - Else
            - Wait for the next ten installations and conduct Test $t + 5$
- If the publisher k passes from all tests
    - Publisher k is legit

$CTIT_{ikn}$: Click-to install time of i^{th} installation from publisher k for campaign n

$Minus$: Number of negative values which are calculated from $CTIT_{ikn} = Install\ Time_{ikn} - Click\ time_{ikn}$

T: the set of tests for publisher k from 1 to $\frac{total\ number\ of\ install\ of\ publisher\ k\ for\ campaign\ n}{sample\ size}$

As mentioned earlier, every installation should be examined one by one for each publisher in each campaign and sub campaign. First of all, a click-to-install time is calculated for each new-coming installation and stored in CTIT(t). This calculation has to be done for every installation. Then it is waited until the number of installations reach the number of sample size to do the testing. Sample size is defined as 10 by the trial and error method. 10 as a sample size is big enough to understand the intention of the publisher and small enough to decide quickly and safely. After the first 10 installations have arrived, first non-parametric sign test is made. As seen in the pseudocode, the first requirement of the sign test is the determination of the sign of the test members. In our procedure, the equation $CTIT_{ikn}^{t} - Time$ is used to define sign of test members. The important matter is defining *Time* to make the decision correctly. Therefore, we observe the distribution of CTIT values for several publishers and note that, if publisher, 70-75% of app installations typically occur during the first hour, 80-85% of installations during the first two hours and 90-95% of installations in the first 24 hours following a click event. Monasteriao (2017) presents the histogram of CTIT values for one-day worth of installations in a non-fraudulent scenario and arrives at the same conclusion that most of the users' installations occur in the first hour after the click.

In the light of these facts, we designed the following sign test for the median of CTIT values of a publisher.

$$\begin{aligned} &H_0: \eta(t) = 7200 \text{ seconds} \\ &H_a: \eta(t) > 7200 \text{ seconds} \end{aligned} \tag{5}$$

In this test, $\eta_0 = 7200$ is adequately large that enables us to safely assume the publisher is fraudulent if the null hypothesis is rejected (i.e. probability of type-I error is small). Hence, *Time* = 7200 seconds. If CTIT is bigger than *Time*, the sign will be positive. If CTIT is less than *Time*, the sign will be negative. If CTIT is equal to *Time*, the sign will

be neutral. For detection of click spamming, only the number of negative ones which is represented with Minus in pseudocode is important because binomial distribution is utilized to calculate the p-value. Cumulative distribution function of binomial distribution is $f(\mathrm{k}, \mathrm{n}, \mathrm{p}) = \mathrm{P}(\mathrm{x} \leq \mathrm{k}) = \sum_{i=0}^{k} \binom{n}{i} p^i (1-p)^{n-i}$ where k represents the number of successes occurring among m trials with probability p. In our procedure, the number of successes is number of negative sign (Minus) and number of trials is equal to 10. In addition, the probability is 0.50000 for nonparametric sing test. Thus, p-value is calculated with the formulation $\sum_{k=0}^{Minus} \binom{10}{k} p^k (1-p)^{10-k}$. For instance, let us assume 5 out of 10 installations have arrived after 2 hours. The p-value is found as 0.62305. After the calculation of the p-value, the value is compared with the significance level α which is 0.05000 in our method. If p-value is less than α, null hypothesis is rejected; otherwise the decision is fail to reject. For example, the p-value which was calculated for 6 installations whose CTIT is greater than 2-hour out of 10 is greater than $\alpha = 0.05000$. Therefore, null hypothesis is failed to reject. If the first test is rejected, the publisher is found as a fraud according to our method of successive runs. In contrast, if the first decision is not a rejection, the second test is applied. After 2nd test, 2 successive reject decisions are needed to accuse the publisher of fraudulence. If 2 reject decisions are not made one after another in the range between 2 and 22 tests, the publisher continues to be investigated. However, after 23rd test (230 installations), blaming publisher for spamming requires 3 successive reject decisions until 433rd test (4330 installations). After that point, 4 reject decisions have to be waited to conclude that a publisher is a fraud or not. table 3.4 represents that number of consecutive installations' range according to the number of tests. If the publisher is figured out to be fraud, it is blocked immediately. For example, let us assume that a publisher is blamed for spamming with 98th test result. It means that 96th, 97th, and 98th null hypothesizes are rejected. Hence, 99th test is unnecessary for that publisher after this point because it is in the list of the blocked spammers. A publisher is tested at most $\frac{The\ Total\ Number\ of\ Installs\ in\ a\ Campaign}{10}$ times. If the publisher passes all tests, the publisher acts legitimate behaviors. In another word, for each null hypothesis that is not rejected, publisher is legit in this campaign. It should not be forgotten that publishers who are found legit in other campaigns can start to commit fraudulent

behaviors after some point. Therefore, it is vital to continue to check legitimacy of publishers in the new campaigns.

4.2 Detection of Click Injection

In click injection, fraudulent publisher's app can detect when other apps are downloaded on the device and trigger clicks right before the installation completes. In this way, they receive undeserved credit for their attributed installations. Click-to-install times of click injecting publishers are typically smaller. However, from the perspective of the advertisers, truncating all installations with small CTIT is not a good idea since not every click that happens shortly before the installation is fraudulent. In this sense, a similar multiple testing procedure can be developed for the detection of injected clicks to prevent advertisers from diverting their budget to fraudsters.

- Start Campaign n
- Set $Install\ Counter = 0$
- Set $Plus = 0$
- For each arriving installation for n^{th} campaign
    - $Install\ Counter = Install\ Counter + 1$
    - $CTIT_{ikn} = Install\ Time_{ikn} - Click\ time_{ikn}$
    - For each test $t \in T$
        - If $Install\ Counter\ \%\ 10 = 0$
            - $Sign = CTIT_{ikn}^{t} - 20$
            - If $Sign > 0$
                - $Plus = Plus + 1$
        - Calculate $Pvalue$ of Test t
        - $Pvalue(t) = \sum_{k=0}^{Plus} \binom{sample\ size}{k} p^k (1-p)^{sample\ size-k}$
        - If $t = 1$
            - If $Pvalue(1) < \alpha$
                - Publisher k is a fraudster
                - Block Publisher k

- Else
  - Wait for the next ten installations and conduct Test 2

- If $t \geq 2$
  - If $Pvalue(t) < \alpha$ & $Pvalue(t+1) < \alpha$
    - Publisher k is a fraudster
    - Block Publisher k
  - Else
    - Wait for the next ten installations and conduct Test $t+2$

- If $t > 22$
  - If $Pvalue(t) < \alpha$ & $Pvalue(t+1) < \alpha$ & $Pvalue(t+2) < \alpha$
    - Publisher k is a fraudster
    - Block Publisher k
  - Else
    - Wait for the next ten installations and conduct Test $t+3$

- If $t > 434$
  - If $Pvalue(t) < \alpha$ & $Pvalue(t+1) < \alpha$ & $Pvalue(t+2) < \alpha$ & $Pvalue(t+3) < \alpha$
    - Publisher k is a fraudster
    - Block Publisher k
  - Else
    - Wait for the next ten installations and conduct Test $t+4$

- If $t > 8524$
  - If $Pvalue(t) < \alpha$ & $Palue(t+1) < \alpha$ & $Pvalue(t+2) < \alpha$ & $Pvalue(t+3) < \alpha$ & $Pvalue(t+4) < \alpha$
    - Publisher k is a fraudster
    - Block Publisher k

- Else
  - Wait for the next ten installations and conduct Test $t + 5$

- If the publisher k passes from all tests
  - Publisher k is legit

$CTIT_{ikn}$: Click-to install time of i^{th} installation from publisher k for campaign n
$Plus$: Number of positive values which are calculated from $CTIT_{ikn} = Install\ Time_{ikn} - Click\ time_{ikn}$
T: number of test for publisher k from 1 to $\frac{total\ number\ of\ install\ of\ publisher\ k\ for\ campaign\ n}{sample\ size}$

Detecting click injection and click spamming is really similar with each other. They are exactly the same about deciding the publisher is legit or fraud. Method of successive runs is utilized to detect click injection exactly like explained in the previous section. Proper number of installations are waited to make tests. The significance level of each test is also $\alpha = 0.05$. Furthermore, fraud is detected according to rules which are represented in table 3.4. The major difference of detection of click injection is hypothesis test from detection of click spamming. The term of click-to-install time represents the time difference between click and the first launch. Download time of mobile application is also included in the click-to-install time. Hence, it is assumed that the total time of downloading, installing and first launching of a mobile app cannot be less than 20 seconds in most cases. Therefore, we designed the following sign test for the median of CTIT values of a publisher.

$$\begin{aligned} &H_0: \eta(t) = 20 \text{ seconds} \\ &H_a: \eta(t) < 20 \text{ seconds} \end{aligned} \tag{6}$$

In this test, $\eta_0 = 20$ is adequately small that enables us to safely assume the publisher is fraudulent if the null hypothesis is rejected. For this type of fraud, positive values that are calculated from $Sign = CTIT_{ikn}^{t} - 20$ is important in terms of calculation of the p-value. The plus term represents total number of positive values from sign equation. If the p-values of all hypothesis tests are greater than 0.05, the publisher is a legitimate

publisher. Even though a publisher is found legit after a campaign, the publisher should continue to be investigated in new campaigns because of possibility of being fraud in the future.

5. EXPERIMENTS AND RESULTS

We will now discuss the results of our experiments conducted with a data set of app installations of a DSP[1] company. The DSP company provides a self-service mobile advertisement platform to its customers (advertisers and/or their agencies) for managing their own ad campaigns. In our experiment, we run tests for approximately 2 million of installations for click spamming and click injection, separately. The method of successive runs was used to detect these frauds. The data is grouped by campaign, sub campaign and publisher. In other words, the installations of each publisher in a sub campaign of a campaign are clustered together in our analysis. For instance, the installations arriving from the publisher P1 of the ad network AN1 which advertises the mobile app of the ad campaign AC1 are grouped together and labeled as the 3-tuple "AC1-AN1-P1". According to this classification, 15263 tuples were identified using the numerical computing software Matlab.

The Matlab codes we have used in our study are shown in Appendix A. The code which is in A.1 is utilized to conduct all possible sign tests of a publisher for click spamming and decide if the publisher is a spammer or not. In this sign test, every installation is examined to make sure whether the click-to-install time of the installation is less than 2-hours. The code conducts the sign test for a sample size of 10 installations of the publisher. Then, the decision is made according to the rule of the code given in A.1. The code which is shown in A.2 is similar to the code shown in A.1. It is used for the sign test of click injection. The only difference is that how the sign test is made. For click injection, every installation is examined to make sure that the installation comes within 20 seconds after the click. If the click-to-install time is less than 20 seconds, the publisher tends to be fraud. Therefore, the sign test of click injection checks the CTIT of each installation to see if it is less than 20 seconds or not. The code given in A.3 counts the positive values

[1] App Samurai Inc., San Francisco, CA, USA

(i.e. the number of installations whose CTIT values are more than 20 seconds) in the sign test of click injection, whereas the code given in A.4 counts the negative values (i.e. the number of installations whose CTIT values are less than 2 hours) in the sign test of click spamming. The code given in A.5 counts the number of rejected tests in both sign tests of click spamming (A.1) and sign test of click injection (A.2) to calculate the total number of rejected tests during the campaign duration. The code given in A.6 clusters the CTITs of all installations belonging to each specific "campaign – sub campaign – publisher" tuple.

The code that is shown is A.7 basically calls all the codes related to click spamming (A1, A4, A5, and A6) for each publisher one by one. Firstly, the code in A.6 is called as a preparation of the sign test. After that the code in A.1 is called to process the collected data. These steps are repeated for every publisher via the code in A.7. Lastly, the code given in A.8 is similar with the code in A.7. The only difference is that the code in A.8 calls all the codes related to click injection (A2, A3, A5, A6).

According to the results that are obtained running these Matlab codes, 1469 publishers out of 15263 publishers are found as click spammers. Click spammers constitute 9.6% of the publishers of the DSP company. The list of the spamming publishers is shown in Appendix B. Table 5.1 provides a small sample of the table given in Appendix B. The "publisher" column in the figure gives the IDs[2] of the publishers who were detected while they were spamming during the campaign duration. The column of "detecting test" illustrates the order of the test in which the publisher is detected as spammer. The column of "total installs" shows the total installations attributed to the publisher during the campaign duration. For example, Publisher 5 (the first publisher in the table) has brought 159 installations during a campaign duration. This publisher has been detected as a spammer at the 9^{th} test. In other words, the publisher is found as spammer after $9 * 10 = 90$ installations have arrived. This means that the click spamming activities of Publisher 5 can only be detected after $(9 * 10)/159 = 56.6\%$ of the total installations have arrived.

[2] These IDs are assigned to the publishes arbitrarily in order to conceal the identity of the publishers due to the company's data privacy protocol.

Table 5.1. The sample of spammers table from Appendix B

Publisher	Detecting Test	Total Installs	Publisher	Detecting Test	Total Installs
5	9	159	2592	68	697
6	4	258	2822	101	1219
10	41	421	2832	7	74
60	5	72	2836	100	1010
65	3	359	2837	1	34
81	1	30	2842	7	1102

Figure 5.1 provides a histogram for the "detecting tests". For example, the bar labeled as "1" in the figure represents the frequency of the publishers who are detected in the first test, while the bar labeled as "10" shows the number of the publishers who are detected in at least 6^{th} test and at most 10^{th} test. Figure 5.1 also shows that it is impossible to detect a spammer in the 2^{nd} test because it is mathematically impossible. According to the rules, after the second test, two successive reject decisions are sought. If the publisher was detected in the 2^{nd} test, the results of the first and second tests would be rejection. However, if the result of the first test was rejection, the publisher would be accused for spamming. For that reason, no one can be detected in the second test. As seen in Figure 5.1, 1385 publishers are detected by the 40^{th} test. In other word, 95% of the click spammers are detected before no more than 400 installations from these publishers have arrived.

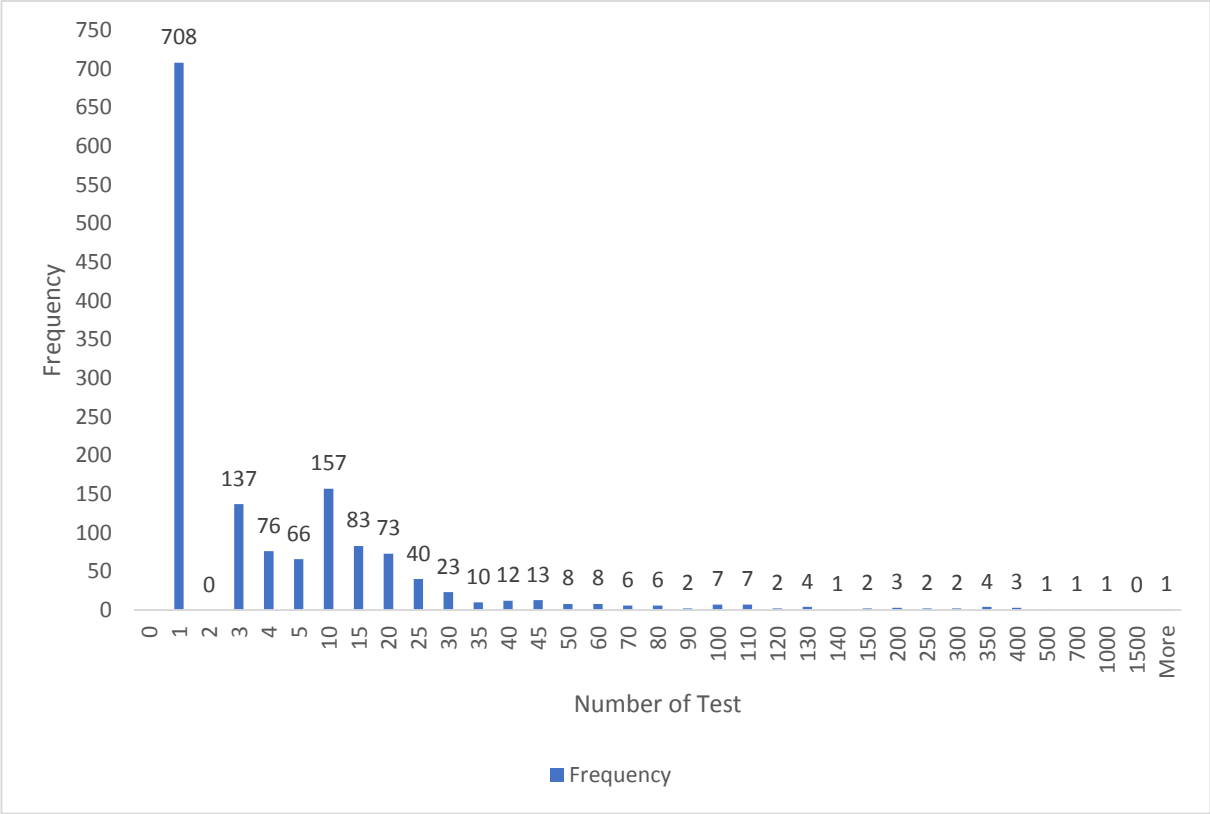

Figure 5.1. The histogram of the "detecting tests"

The distribution of the means of the click-to-install time of the fraudulent publishers is shown at the top graph of Figure 5.2. The spammers whose mean CTIT value is less than 2 hours composes only 2% of the spamming publishers. This small group of spammers has a mean CTIT value less than 2 hours because they most probably mix up their legit advertising with occasional spamming in a random fashion. Most of the click spammers have a mean value that is more than one day. The bottom graph in Figure 5.2 illustrates distribution of mean of CTIT of legit publishers. As mentioned in Chapter 3, the mean of CTITs can be high because of some legitimate reasons. For example, some users may have forgotten to launch the application. This graph supports the claim that the mean is not a reliable descriptive statistic to understand the click spamming.

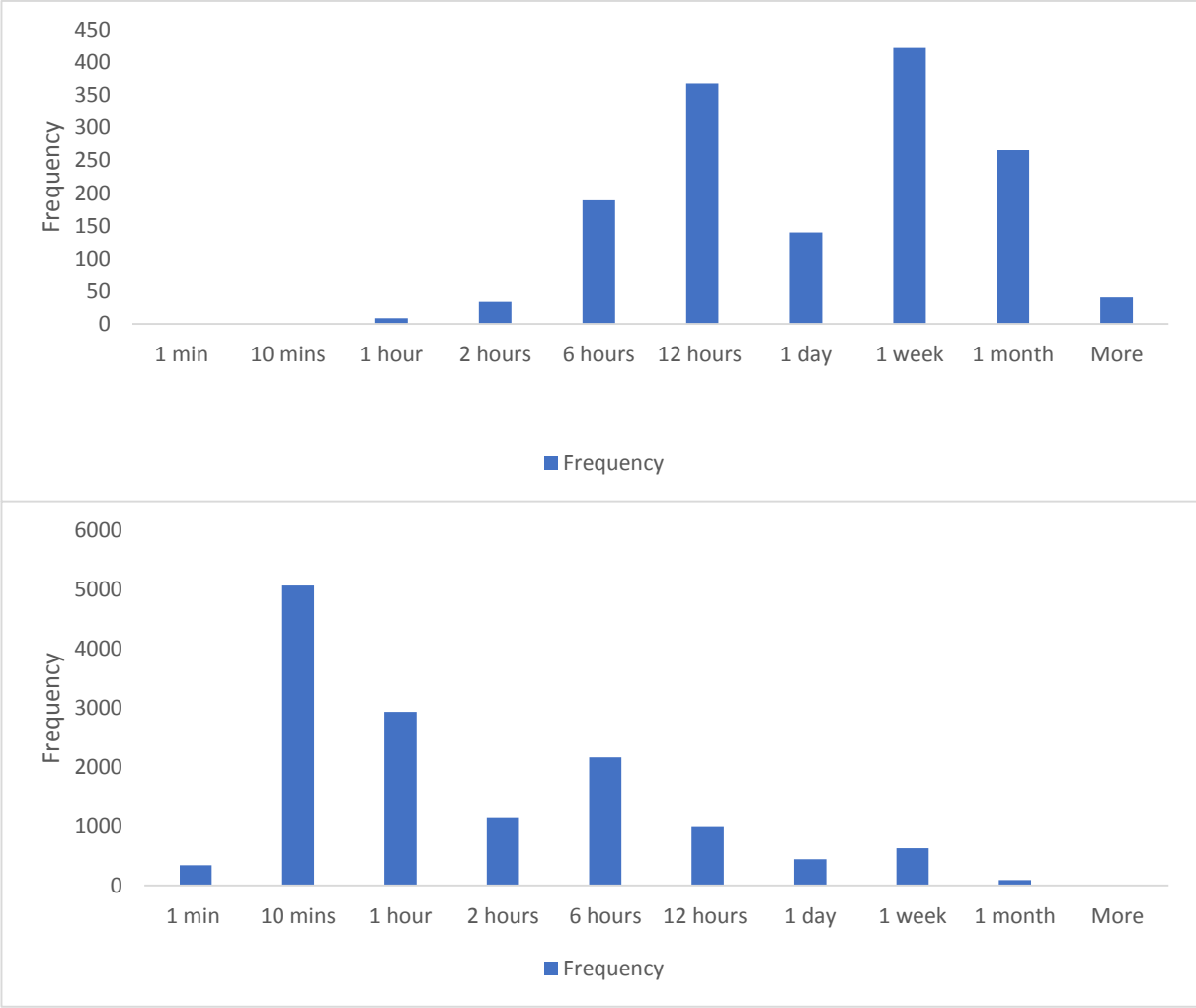

Figure 5.2. The histograms of mean of CTIT values for the fraudulent publishers (top) and the legit publishers (bottom)

The median value of click-to-install times, on the other hand, provides a clear understanding for legit and spamming publishers. As seen in the bottom graph of Figure 5.3, 90% of the legit publishers have median values that are less than 2 hours. In the top graph of Figure 5.3, even though most of the spammers have large median values, 20% of the fraudulent publishers have median values that are less than 2 hours. These publishers support the result of our analysis which is mentioned in Chapter 3. If the tests had been applied for the all-time installations of publishers, 303 spammers would have passed the tests. Financial loss is prevented with our multiple testing procedure.

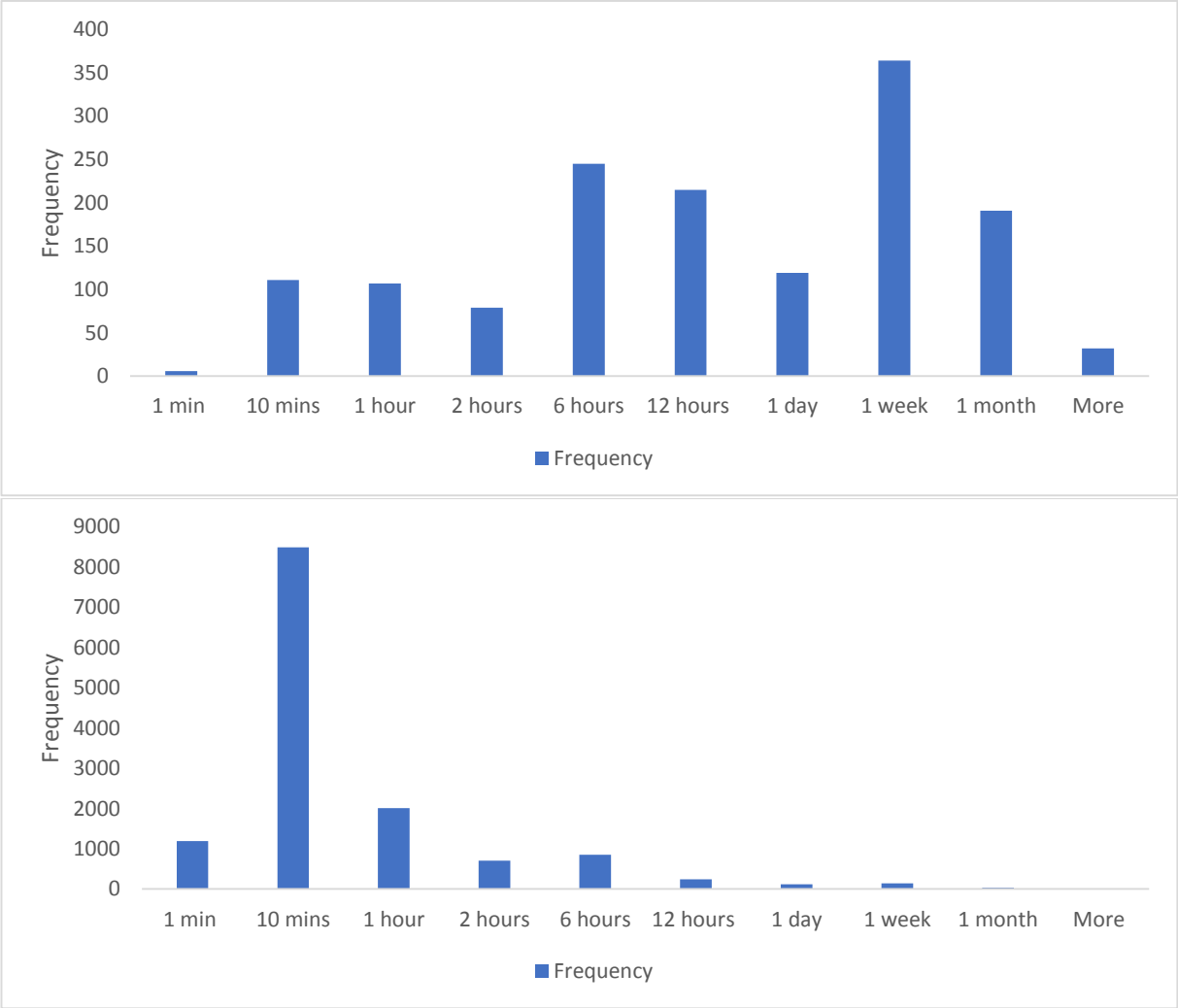

Figure 5.3. The histograms of median of CTIT values for the fraudulent publishers (top) and the legit publishers (bottom)

Click injection is a fraudulent activity which is used much less than click spamming. The main reason for underutilization of click injection is that click injection requires an untraceable spy app which can detect an installation just before it is executed and it is technically more difficulty than click spamming. In order to detect click injection, we applied the procedure discussed in Chapter 4. According to the results of our experiment, five publishers (out of 15263 publishers) is detected as fraudsters who conduct click injection. Table 5.2 shows the list of these publishers. Each publisher is represented with a row in the table. The first column provides publisher ID. The column of "detecting test" specifies the order of the first test indicating a click injection activity. The column of "total installs" shows the total number of installations of the detected publisher during the campaign duration. The last column shows the total number of rejected tests during the

campaign. In the best case (Publisher 6551), the click injection is detected in the 1st test. The fraudster is detected before $(1 * 10)/158 = 6\%$ of the total installations have arrived. In the worst case, click injection is not detected until the last installation arrives. The number of rejects of the publisher 6550 is also worth mentioning. Although some reject decisions had been made before the publisher was blamed for click injection, publisher was accused of injection in 15th test because of the rule of our multiple testing procedure. The method of successive runs prevents the publishers from being wrongly accused.

Although only 5 publishers are detected as fraudster who make click injection, we believe that in reality there are more publishers who are engaged in click injection. Unfortunately, the DSP company does not have a record on click injection. Therefore, we cannot calculate our type-II error for the click injection experiment. The reason for having a possibly high type-II error rate with the method of successive runs to detect click injection is that the CTIT values less than 20 seconds are much more stronger indicator for (injection) fraud than CTIT values more than 2 hours for (spamming) fraud.

Table 5.2 The list of fraudulent publishers who made click injection

Publisher	Detecting test	Total Installs	Number of Rejects
591	1	10	1
592	1	10	1
6550	15	622	23
6551	1	158	5
13606	1	28	1

While we detected 0.34% of the publishers as click injecting fraudsters, we found that the click spamming ones constitute 10% of all the publishers. This finding supports the fact that click spamming is a more popular fraud type than click injection. Our findings also suggest that there is no publisher who conducts click spamming and click injection together.

6. CONCLUSION

Click spamming and click injection are common types of frauds in mobile display advertising. However, the fraudulent activities of fraudsters can still be detected by applying some statistical techniques on click-to-install time (CTIT) values. In this study, we have defined a novel multiple testing procedure which conducts sign tests on CTIT values of a publisher periodically. If we observe r successive rejections in these tests, where r is a function of the number of tests conducted so far, we tag the publisher as fraudulent. Otherwise, publisher passes the testing procedure and is concluded to be legit. This technique can be utilized to detected both click spamming and click injection activities with a small adjustment which is merely changing the null hypothesis. We have also showed that our procedure has a false-positive error rate of at most $\bar{\alpha} = 0.05$. Lastly, we run an experiment with 15263 publishers. According to the results of the experiment, we tag 1469 publishers for click spamming and 5 publishers for click injection.

As a future research direction, our multiple testing procedures can be tested with a set of mobile advertisements campaigns *in real time* and the rate of success for our methods can be calculated. The method of successive runs applied in real time to detect click spamming and click injection can protect advertisers from further financial losses. According to the results of the click injection experiment, the number of click injecting publishers is less than the expected level. Therefore, deciding rule for click injection in the method of successive runs can be tuned up to reduce type-II error rate and achieve more accurate results.

Another future research opportunity related to click injection is about η_0 value in Equation (2). The value of η_0 is assumed to be a lower bound for CTIT values of publishers which do not conduct any click injection activity. Since the size (in megabytes) of the mobile application is directly proportion with the download time of the app, it immediately affects the click-to-install time. Therefore, η_0 can be expressed as a function

of app size. Expressing η_0 as an appropriate function of app size in the future can provide a more accurate detection of click-injecting publishers.

REFERENCES

Badhe, A. (2016). Click Fraud Detection In Mobile Ads Served *In Programmatic Exchanges. International Journal of Scientific Technology & Research*, 5(4), 1.

Bland, J. M., & Altman, D. G. (1995). Multiple significance tests: the Bonferroni method. *BMJ*, 310(6973), 170.

Blizard, T., & Livic, N. (2012). Click-fraud monetizing malware: A survey and case study. *Proceedings of 7th International Conference on Malicious and Unwanted Software (MALWARE 2012)* (pp. 67-72).

Cho, G., Cho, J., Song, Y., & Kim, H. (2015). An empirical study of click fraud in mobile advertising networks. *In Proceedings of 10th International Conference on Availability, Reliability and Security (ARES)* (pp. 382-388).

Cho, G., Cho, J., Song, Y., Choi, D., & Kim, H. (2016). Combating online fraud attacks in mobile-based advertising. *EURASIP Journal on Information Security*, (1)2.

Crussell, J., Stevens, R., & Chen, H. (2014). Madfraud: Investigating ad fraud in android applications. *Proceedings of the 12th annual international conference on Mobile systems, applications, and services* (pp. 123-134).

Daswani, N., Mysen, C., Rao, V., Weis, S., Gharachorloo, K., & Ghosemajumder, S. (2008). Online advertising fraud. *Crimeware: understanding new attacks and defenses*, 40(2), 1-28.

Dave, V., Guha, S., & Zhang, Y. (2012). Measuring and fingerprinting click-spam in ad networks. *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication* (pp. 175-186).

Feller, W. (1968). *An Introduction to Probability Theory and Its Applications*, Vol. 1. (3rd ed.). New York: J. Wiley & Sons.

Gupta, A., Tiwari, A., Venkatraman, G., Cheung, D., Bennett, S. R., & Koen, D. B. (2014). U.S. Patent No. 8,799,069. Washington, DC: U.S. Patent and Trademark Office.

Hill, A. T., Vandermay, J., & Error, B. M. (2014). U.S. Patent No. 8,880,541. Washington, DC: U.S. Patent and Trademark Office

Holm, S. (1979). A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics*, 65-70.

Immorlica, N., Jain, K., Mahdian, M., & Talwar, K. (2005). Click fraud resistant methods for learning click-through rates. *International Workshop on Internet and Network Economics* (pp. 34-45). Springer, Berlin, Heidelberg.

Iqbal, M. S., Zulkernine, M., Jaafar, F., & Gu, Y. (2018). Protecting Internet users from becoming victimized attackers of click-fraud. *Journal of Software: Evolution and Process*, 30(3), e1871.

Jain, K., & Talwar, K. (2007). U.S. Patent Application No. 11/178,528.

Kitts, B. J., Najm, T., & Burdick, B. (2008). U.S. Patent Application No. 11/745,264.

Linden, J., & Teeter, T. (2012). U.S. Patent No. 8,321,269. Washington, DC: U.S. Patent and Trademark Office.

Liu, B., Nath, S., Govindan, R., & Liu, J. (2014). DECAF: Detecting and Characterizing Ad Fraud in Mobile Apps. *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI '14)* (pp. 57-70).

Monasterio, J. D. (2017). Tagging Click-Spamming Suspicious Installs in Mobile Advertising Through Time Delta Distributions. *Proceedings of Simposio Argentino de GRANdes DAtos (AGRANDA)-JAIIO 46* (Córdoba, 2017).

Nieborg, D. B. (2016). App advertising: The rise of the player commodity. In J. F. Hamilton, R. Bodle, & E. Korin (Eds.), *Explorations in critical studies of advertising* (pp. 28–41). New York: Routledge.

Oentaryo, R., Lim, E. P., Finegold, M., Lo, D., Zhu, F., Phua, C., & Perera, K. (2014). Detecting click fraud in online advertising: a data mining approach. *The Journal of Machine Learning Research*, 15(1), 99-140.

Perera, K. S., Neupane, B., Faisal, M. A., Aung, Z., & Woon, W. L. (2013). A novel ensemble learning-based approach for click fraud detection in mobile advertising. *In Mining Intelligence and Knowledge Exploration* (pp. 370-382). Springer, Cham.

Shields, R. (2016). REPORT: Ad fraud is 'second only to the drugs trade' as a source of income for organized crime. *Business Insider*. Retrieved from http://www.businessinsider.com/wfa-report-ad-fraud-will-cost-advertisers-50-billion-by-2025-2016-6.

Šidák, Z. (1967). Rectangular confidence regions for the means of multivariate normal distributions. *Journal of the American Statistical Association*, 62(318), 626-633.

Smith, B., Lester, C., & Karrels, E. L. (2011). U.S. Patent No. US7933984B1. Washington, DC: U.S. Patent and Trademark Office.

Soubusta, S. (2008). On Click Fraud. *Information Wissenschaft und Praxis*, 59(2), 136

Sprent P. (1989). *Applied Nonparametric Statistical Methods*. (4th ed.). London: Chapman and Hall.

Springborn, K., & Barford, P. (2013, August). Impression Fraud in On-line Advertising via Pay-Per-View Networks. *In USENIX Security Symposium* (pp. 211-226).

Zingirian, N., & Benini, M. (2018). Click Spam Prevention Model for On-Line Advertisement. Manuscript submitted for publication. Retrieved from http://arxiv.org/abs/1802.02480

APPENDIX A

A.1 Matlab Code for Sign Test for Click Spamming

```
%sign test for spamming
function [decision, numReject, ordReject]= sign_test(array)
%decision="a";
ordReject=0;
sz = size(array);
szArray=zeros(sz(1),1);
%Sign Array s
for i=1:1:sz(1)
if array(i,1)-7200 >0
    szArray(i,1)=1;
elseif array(i,1)-7200 <0
    szArray(i,1) =-1;
else
    szArray(i,1) =0;
end

end
%P_Value
row=floor(sz(1)/10);
pValues= zeros(row,1);
k=1;
son=sz(1)-mod(sz(1),10);
for i=1:10:son-9
    b=szArray(i:i+9);

        ones=countOnes(b);
        mones=countMOnes(b);
        count=ones+mones;
        %small=min(b);
        pValues(k,1)=binocdf(mones,count,0.5);
        k=k+1;



end
%Comparison of Values with 0.05  0=reject, 1=fail to reject
sonuc=zeros(row,1);
for i=1:1:sz(1)/10
if pValues(i,1)<0.05
```

```
    sonuc(i,1)=0;
else
    sonuc(i,1)=1;
end

end
counter=1;
sonucSz=size(sonuc);
numReject= countReject(sonuc);
if sonucSz(1)<=1

        if sonuc(1,1)==0
          decision="fraud";
          ordReject=counter;
        else
          decision="legit";

        end

elseif sonucSz(1)>1 && sonucSz(1)<=22
    if sonuc(1,1)==0
          decision="fraud";
          ordReject=counter;
          return
    end
          counter=counter+1;


        if sonucSz(1)==22
            for i=2:1:21
                if sonuc(i,1)==0 && sonuc(i+1,1)==0
                decision="fraud";
                ordReject=counter+1;
                return
                else
                decision="legit";
                end
                counter=counter+1;
            end


        else
            if sonucSz(1)==2
            decision="legit";
             return
            else
        for i=2:1:sonucSz(1)-1
            if sonuc(i,1)==0 && sonuc(i+1,1)==0
             decision="fraud";
```

```
                ordReject=counter+1;
                return
                else
                decision="legit";
                end
                counter=counter+1;
            end
                end
            end




elseif sonucSz(1)>22 && sonucSz(1)<=434
    if sonuc(1,1)==0
          decision="fraud";
          ordReject=counter;
          return
    end
          counter=counter+1;

    for i=2:1:21
        if sonuc(i,1)==0 && sonuc(i+1,1)==0
          decision="fraud";
          ordReject=counter+1;
          return
        end
        counter=counter+1;
    end

    if sonucSz(1)==434
        for i=22:1:432
            if sonuc(i,1)==0 && sonuc(i+1,1)==0 &&
sonuc(i+2,1)==0
            decision="fraud";
            ordReject=counter+2;
            return
            else
                decision="legit";
            end
            counter=counter+1;
        end

    else
        if sonucSz(1)== 23
            decision="legit";
            return
        else
```

```
        for i=22:1:sonucSz(1)-2
        if sonuc(i,1)==0 && sonuc(i+1,1)==0 &&
sonuc(i+2,1)==0
          decision="fraud";
          ordReject=counter+2;
          return
        else
            decision="legit";
        end
        counter=counter+1;
        end
        end
    end

else
    if sonuc(1,1)==0
          decision="fraud";
          ordReject=counter;
          return
     end
          counter=counter+1;

    for i=2:1:21
        if sonuc(i,1)==0 && sonuc(i+1,1)==0
          decision="fraud";
          ordReject=counter+1;
          return
        end
        counter=counter+1;
    end
    for i=22:1:432
        if sonuc(i,1)==0 && sonuc(i+1)==0 && sonuc(i+2)==0
          decision="fraud";
          ordReject=counter+2;
          return
        end
        counter=counter+1;
    end

        if sonucSz(1)==8524
    for i=431:1:8521
        if sonuc(i,1)==0 && sonuc(i+1,1)==0 &&
sonuc(i+2,1)==0 && sonuc(i+3,1)==0
          decision="fraud";
          ordReject=counter+2;
          return
        else
            decision="legit";
        end
```

```
        counter=counter+1;
    end

    else
        if sonucSz(1)==435
            decision="legit";
            return
        else
        for i=432:1:sonucSz(1)-3
        if sonuc(i,1)==0 && sonuc(i+1,1)==0 &&
sonuc(i+2,1)==0 && sonuc(i+3,1)==0
          decision="fraud";
          ordReject=counter+2;
          return
        else
            decision="legit";
        end
        counter=counter+1;
        end
        end
        end


end

end
```

A.2 Matlab Code for Sign Test for Click Injection

```
%sign test for injection
function [decision, numReject, ordReject]=
Psign_test(array)
%decision="a";
ordReject=0;
sz = size(array);
szArray=zeros(sz(1),1);
%Sign Array
for i=1:1:sz(1)
if array(i,1)-20 >0
    szArray(i,1)=1;
elseif array(i,1)-20 <0
    szArray(i,1) =-1;
else
    szArray(i,1) =0;
end

end
%P_Value
```

```
row=floor(sz(1)/10);
pValues= zeros(row,1);
k=1;
son=sz(1)-mod(sz(1),10);
for i=1:10:son-9
    b=szArray(i:i+9);

        ones=countOnes(b);
        mones=countMOnes(b);
        count=ones+mones;
        %small=min(b);
        pValues(k,1)=binocdf(ones,count,0.5);
        k=k+1;



end
% Comparison of Values with 0.05  0=reject, 1=fail to
reject
sonuc=zeros(row,1);
for i=1:1:sz(1)/10
if pValues(i,1)<0.05
    sonuc(i,1)=0;
else
    sonuc(i,1)=1;
end

end
counter=1;
sonucSz=size(sonuc);
numReject= countReject(sonuc);
if sonucSz(1)<=1

        if sonuc(1,1)==0
          decision="fraud";
          ordReject=counter;
        else
          decision="legit";

        end

elseif sonucSz(1)>1 && sonucSz(1)<=22
    if sonuc(1,1)==0
          decision="fraud";
          ordReject=counter;
          return
    end
          counter=counter+1;
```

```
        if sonucSz(1)==22
            for i=2:1:21
                if sonuc(i,1)==0 && sonuc(i+1,1)==0
                decision="fraud";
                ordReject=counter+1;
                return
                else
                decision="legit";
                end
                counter=counter+1;
            end


        else
            if sonucSz(1)==2
            decision="legit";
             return
            else
        for i=2:1:sonucSz(1)-1
            if sonuc(i,1)==0 && sonuc(i+1,1)==0
             decision="fraud";
             ordReject=counter+1;
            return
            else
            decision="legit";
            end
            counter=counter+1;
        end
            end
        end




elseif sonucSz(1)>22 && sonucSz(1)<=434
    if sonuc(1,1)==0
          decision="fraud";
          ordReject=counter;
          return
    end
          counter=counter+1;

    for i=2:1:21
        if sonuc(i,1)==0 && sonuc(i+1,1)==0
          decision="fraud";
          ordReject=counter+1;
          return
```

```
        end
        counter=counter+1;
    end

    if sonucSz(1)==434
        for i=22:1:432
            if sonuc(i,1)==0 && sonuc(i+1,1)==0 &&
sonuc(i+2,1)==0
            decision="fraud";
            ordReject=counter+2;
            return
            else
                decision="legit";
            end
            counter=counter+1;
        end

    else
        if sonucSz(1)== 23
            decision="legit";
            return
        else
        for i=22:1:sonucSz(1)-2
        if sonuc(i,1)==0 && sonuc(i+1,1)==0 &&
sonuc(i+2,1)==0
          decision="fraud";
          ordReject=counter+2;
          return
        else
            decision="legit";
        end
        counter=counter+1;
        end
        end
    end

else
    if sonuc(1,1)==0
          decision="fraud";
          ordReject=counter;
          return
     end
          counter=counter+1;

    for i=2:1:21
        if sonuc(i,1)==0 && sonuc(i+1,1)==0
          decision="fraud";
          ordReject=counter+1;
          return
```

```
            end
            counter=counter+1;
        end
        for i=22:1:432
            if sonuc(i,1)==0 && sonuc(i+1)==0 && sonuc(i+2)==0
              decision="fraud";
              ordReject=counter+2;
              return
            end
            counter=counter+1;
        end

            if sonucSz(1)==8524
        for i=431:1:8521
            if sonuc(i,1)==0 && sonuc(i+1,1)==0 &&
sonuc(i+2,1)==0 && sonuc(i+3,1)==0
              decision="fraud";
              ordReject=counter+2;
              return
            else
                decision="legit";
            end
            counter=counter+1;
        end

        else
            if sonucSz(1)==435
                decision="legit";
                return
            else
            for i=432:1:sonucSz(1)-3
            if sonuc(i,1)==0 && sonuc(i+1,1)==0 &&
sonuc(i+2,1)==0 && sonuc(i+3,1)==0
              decision="fraud";
              ordReject=counter+2;
              return
            else
                decision="legit";
            end
            counter=counter+1;
            end
            end
            end


end

end
```

A.3 Matlab Code for Counting Positive Values for Sign Test

```
function [x]= countOnes(array)
%number of ones in array
sz = size(array);
n=0;
for i=1:1:sz(1)
    if array(i) == 1
        n=n+1;
    end
x = n;

end
```

A.4 Matlab Code for Counting Negative Values for Sign Test

```
function [x]= countMOnes(array)
%umber of minus ones in array
sz = size(array);
n=0;
for i=1:1:sz(1)
    if array(i) == -1
        n=n+1;
    end
x = n;

end
```

A.5 Matlab Code for Counting Rejected Test

```
function [x]= countReject(array)
%arraydeki bir sayýsý
sz = size(array);
n=0;
for i=1:1:sz(1)
    if array(i) == 0
        n=n+1;
    end
x = n;

end
```

A.6 Matlab Code for Collecting Data to Do Sign Test for a Publisher

```
%Function for collecting data from excel files
```

```
function [array, duration]= collectdata(camp, subcamp, pub,
numInst, data1, data2)
array = zeros(numInst,1);

%[~,~,data1]=xlsread('data1','duz');
%[~,~,data2]=xlsread('data2','duz');
k=1;
%disp('data okundu');

endpoint1 = size(data1);
endpoint2 = size(data2);
strcamp = ischar(camp);
if strcamp==1
    camp = convertCharsToStrings(camp);
end

strsubcamp = ischar(subcamp);
if strsubcamp==1
    subcamp= convertCharsToStrings(subcamp);
end
strpub = ischar(pub);
if strpub==1
    pub=convertCharsToStrings(pub);
end
%disp(strcamp);
%disp(strsubcamp);
%disp(strpub);



if strcamp==1 && strsubcamp==1 && strpub==1
    for i=1:1:endpoint1(1)-1
% x for camp, y for subcamp, z for pub
        x=strcmp(convertCharsToStrings(data1{i+1,3}),camp);
        y=strcmp(convertCharsToStrings(data1{i+1,4}),subcam
p);
        z=strcmp(convertCharsToStrings(data1{i+1,5}),pub);
        if x== 1 && y==1 && z==1
        array(k,1)= data1{i+1,1};
        array(k,2)= data1{i+1,6};
        k=k+1;
        end
    end
elseif strcamp==0 && strsubcamp==1 && strpub==1
    for i=1:1:endpoint1(1)-1
% x for camp, y for subcamp, z for pub
%         x=strcmp(data1{i+1,3},camp);
        y=strcmp(convertCharsToStrings(data1{i+1,4}),subcam
p);
```

```
        z=strcmp(convertCharsToStrings(data1{i+1,5}),pub);
        if data1{i+1,3}== camp && y==1 && z==1
        array(k,1)= data1{i+1,1};
        array(k,2)= data1{i+1,6};
        k=k+1;
        end
    end
elseif strcamp==1 && strsubcamp==0 && strpub==1
    for i=1:1:endpoint1(1)-1
% x for camp, y for subcamp, z for pub
        x=strcmp(convertCharsToStrings(data1{i+1,3}),camp);
%         y=strcmp(data1{i+1,4},subcamp);
        z=strcmp(convertCharsToStrings(data1{i+1,5}),pub);
        if x==1 && data1{i+1,4}==subcamp && z==1
        array(k,1)= data1{i+1,1};
        array(k,2)= data1{i+1,6};
        k=k+1;
        end
    end
elseif strcamp==1 && strsubcamp==1 && strpub==0
    for i=1:1:endpoint1(1)-1
% x for camp, y for subcamp, z for pub
        x=strcmp(convertCharsToStrings(data1{i+1,3}),camp);
        y=strcmp(convertCharsToStrings(data1{i+1,4}),subcam
p);
%         z=strcmp(data1{i+1,5},pub);
        if x==1 && y==1 && data1{i+1,5}==pub
        array(k,1)= data1{i+1,1};
        array(k,2)= data1{i+1,6};
        k=k+1;
        end
    end
elseif strcamp==1 && strsubcamp==0 && strpub==0
    for i=1:1:endpoint1(1)-1
% x for camp, y for subcamp, z for pub
        x=strcmp(convertCharsToStrings(data1{i+1,3}),camp);
%         y=strcmp(data1{i+1,4},subcamp);
%         z=strcmp(data1{i+1,5},pub);
        if x==1 && data1{i+1,4}==subcamp &&
data1{i+1,5}==pub
        array(k,1)= data1{i+1,1};
        array(k,2)= data1{i+1,6};
        k=k+1;
        end
    end
elseif strcamp==0 && strsubcamp==1 && strpub==0
    for i=1:1:endpoint1(1)-1
% x for camp, y for subcamp, z for pub
%         x=strcmp(data1{i+1,3},camp);
```

```
        y=strcmp(convertCharsToStrings(data1{i+1,4}),subcam
p);
%         z=strcmp(data1{i+1,5},pub);
        if data1{i+1,3}== camp && y==1 && data1{i+1,5}==pub
        array(k,1)= data1{i+1,1};
        array(k,2)= data1{i+1,6};
        k=k+1;
        end
    end
elseif strcamp==0 && strsubcamp==0 && strpub==1
    for i=1:1:endpoint1(1)-1
% x for camp, y for subcamp, z for pub
%         x=strcmp(data1{i+1,3},camp);
%         y=strcmp(data1{i+1,4},subcamp);
        z=strcmp(convertCharsToStrings(data1{i+1,5}),pub);
        if data1{i+1,3}== camp && data1{i+1,4}==subcamp &&
z==1
        array(k,1)= data1{i+1,1};
        array(k,2)= data1{i+1,6};
        k=k+1;
        end
    end
else
    for i=1:1:endpoint1(1)-1
        if data1{i+1,3}==camp & data1{i+1,4}==subcamp &
data1{i+1,5}==pub
            array(k,1)= data1{i+1,1};
            array(k,2)= data1{i+1,6};
            k=k+1;
        end
    end
end


if strcamp==1 && strsubcamp==1 && strpub==1
    for i=1:1:endpoint2(1)-1
% x for camp, y for subcamp, z for pub
        x=strcmp(convertCharsToStrings(data2{i+1,3}),camp);
        y=strcmp(convertCharsToStrings(data2{i+1,4}),subcam
p);
        z=strcmp(convertCharsToStrings(data2{i+1,5}),pub);
        if x== 1 && y==1 && z==1
        array(k,1)= data2{i+1,1};
        array(k,2)= data2{i+1,6};
        k=k+1;
        end
    end
elseif strcamp==0 && strsubcamp==1 && strpub==1
    for i=1:1:endpoint2(1)-1
```

```
% x for camp, y for subcamp, z for pub
%         x=strcmp(data2{i+1,3},camp);
        y=strcmp(convertCharsToStrings(data2{i+1,4}),subcam
p);
        z=strcmp(convertCharsToStrings(data2{i+1,5}),pub);
        if data2{i+1,3}== camp && y==1 && z==1
        array(k,1)= data2{i+1,1};
        array(k,2)= data2{i+1,6};
        k=k+1;
        end
    end
elseif strcamp==1 && strsubcamp==0 && strpub==1
    for i=1:1: endpoint2(1)-1
% x for camp, y for subcamp, z for pub
        x=strcmp(convertCharsToStrings(data2{i+1,3}),
camp);
%         y=strcmp(data2{i+1,4}, subcamp);
        z=strcmp(convertCharsToStrings(data2{i+1,5}), pub);
        if x==1 && data2{i+1,4} ==subcamp && z==1
        array(k,1)= data2{i+1,1};
        array(k,2)= data2{i+1,6};
        k=k+1;
        end
    end
elseif strcamp==1 && strsubcamp==1 && strpub==0
    for i=1:1: endpoint2(1)-1
% x for camp, y for subcamp, z for pub
        x=strcmp(convertCharsToStrings(data2{i+1,3}),
camp);
        y=strcmp(convertCharsToStrings(data2{i+1,4}),
subcamp);
%         z=strcmp(data2{i+1,5}, pub);
        if x==1 && y==1 && data2{i+1,5} ==pub
        array(k,1)= data2{i+1,1};
        array(k,2)= data2{i+1,6};
        k=k+1;
        end
    end
elseif strcamp==1 && strsubcamp==0 && strpub==0
    for i=1:1: endpoint2(1)-1
% x for camp, y for subcamp, z for pub
        x=strcmp(convertCharsToStrings(data2{i+1,3}),camp);
%         y=strcmp(data2{i+1,4},subcamp);
%         z=strcmp(data2{i+1,5},pub);
        if x==1 && data2{i+1,4}==subcamp &&
data2{i+1,5}==pub
        array(k,1)= data2{i+1,1};
        array(k,2)= data2{i+1,6};
        k=k+1;
```

```
        end
    end
elseif strcamp==0 && strsubcamp==1 && strpub==0
    for i=1:1: endpoint2(1)-1
% x for camp, y for subcamp, z for pub
%        x=strcmp(data2{i+1,3}, camp);
        y=strcmp(convertCharsToStrings(data2{i+1,4}),subcam
p);
%        z=strcmp(data2{i+1,5}, pub);
        if data2{i+1,3} == camp && y==1 && data2{i+1,5} ==
pub
        array(k,1) = data2{i+1,1};
        array(k,2) = data2{i+1,6};
        k=k+1;
        end
    end
elseif strcamp==0 && strsubcamp==0 && strpub==1
    for i=1:1:endpoint2(1)-1
% x for camp, y for subcamp, z for pub
%        x=strcmp(data2{i+1,3}, camp);
%        y=strcmp(data2{i+1,4}, subcamp);
        z=strcmp(convertCharsToStrings(data2{i+1,5}), pub);
        if data2{i+1,3}== camp && data2{i+1,4}==subcamp &&
z==1
        array(k,1)= data2{i+1,1};
        array(k,2)= data2{i+1,6};
        k=k+1;
        end
    end
else
    for i=1:1: endpoint2(1)-1
        if data2{i+1,3} == camp & data2{i+1,4}==subcamp &
data2{i+1,5}==pub
            array(k,1) = data2{i+1,1};
            array(k,2) = data2{i+1,6};
            k=k+1;
        end
    end
end

array=sortrows(array);
duration = (array(numInst,1)-array(1,1))/(60*60*24);
end
```

A.7 Matlab Code to Detect Click Spamming

```
[~,~,data1]=xlsread('data1','duz');
[~,~,data2]=xlsread('data2','duz');
```

```
spamming={0};
for a=1:1:15244
    for b=1:1:10
    spamming{a,b}=0;
    end
end
[~,~,veri]=xlsread('Tez Descriptive
Statistics1','Publishers');
spamming{1,10}='Decision';
spamming{1,1}='Campaign ID';
spamming{1,2}='Subcampaign ID';
spamming{1,3}='Publisher Number';
spamming{1,4}='Number of Total Installs';
spamming{1,5}='Mean';
spamming{1,6}='Median';
spamming{1,7}='Campaign Duration';
spamming{1,8}='Spamming Order';
spamming{1,9}='Number of Reject';

for i=2:1:15244
    display(i);
    [array, duration]= collectdata(veri{i,1}, veri{i,2},
veri{i,3}, veri{i,4}, data1, data2);
    for j=1:1:6
        spamming{i,j}=veri{i,j};
    end
    spamming{i,7}=duration;
    endpoint=size(array);
    array1=zeros(endpoint(1),1);
    for k=1:1:endpoint
        array1(k,1)=array(k,2);
    end


    [decision, numReject, ordReject]= sign_test(array1);
    spamming{i,8}=ordReject;
    spamming{i,9}=numReject;
    spamming{i,10}=decision;
end
```

A.8 Matlab Code to Detect Click Injection

```
[~,~,data1]=xlsread('data1','duz');
[~,~,data2]=xlsread('data2','duz');
injection={0};
for a=1:1:15244
    for b=1:1:10
    injection{a,b}=0;
```

```
    end
end
[~,~,veri]=xlsread('Tez Descriptive
Statistics1','Publishers');
injection{1,10}='Decision';
injection{1,1}='Campaign ID';
injection{1,2}='Subcampaign ID';
injection{1,3}='Publisher Number';
injection{1,4}='Number of Total Installs';
injection{1,5}='Mean';
injection{1,6}='Median';
injection{1,7}='Campaign Duration';
injection{1,8}='Spamming Order';
injection{1,9}='Number of Reject';

for i=2:1:15244
    display(i);
    [array, duration]= collectdata(veri{i,1}, veri{i,2},
veri{i,3}, veri{i,4}, data1, data2);
    for j=1:1:6
        injection{i,j}=veri{i,j};
    end
    injection{i,7}=duration;
    endpoint=size(array);
    array1=zeros(endpoint(1),1);
    for k=1:1:endpoint
        array1(k,1)=array(k,2);
    end


    [decision, numReject, ordReject]= Psign_test(array1);
    injection{i,8}=ordReject;
    injection{i,9}=numReject;
    injection{i,10}=decision;
end
```

APPENDIX B

B.1 Detected Click Spammers According to Experiment

Publisher	Detecting Test	Total Installs	Publisher	Detecting Test	Total Installs	Publisher	Detecting Test	Total Installs	Publisher	Detecting Test	Total Installs
5	9	159	2592	68	697	7542	1	23	11783	1	10
6	4	258	2822	101	1219	7543	1	39	11785	16	396
10	41	421	2832	7	74	7593	1	19	11786	3	227
60	5	72	2836	100	1010	7594	1	14	11790	3	62
65	3	359	2837	1	34	7608	1	10	11824	1	31
81	1	30	2842	7	1102	7615	40	757	11825	1	44
152	1	20	2910	12	121	7619	327	5692	11826	1	27
158	1	16	2923	7	70	7627	34	406	11830	1	20
159	1	115	3111	17	728	7661	28	343	11831	3	211
160	1	25	3157	1	25	7664	33	423	11868	3	72
161	3	362	3175	17	303	7668	68	820	11897	1	13
162	1	78	3193	5	54	7683	116	1160	11908	15	151
163	3	701	3195	3	31	7691	47	1244	12163	20	294
165	1	11	3196	5	86	7694	29	369	12165	8	184
167	1	13	3358	1	17	7702	20	223	12166	56	4343
169	1	65	3360	1	24	7711	383	9670	12168	8	95
170	1	18	3467	1	184	7733	3	68	12170	9	121
171	1	12	3469	1	29	7736	1	13	12244	1	14
172	3	151	3473	1	406	7739	10	109	12346	51	609
173	1	21	3475	1	12	7745	1	51	12347	35	389
174	1	10	3476	47	978	7749	1	12	12354	20	1738
175	1	10	3485	4	130	7750	6	79	12355	1	150
177	1	25	3488	9	136	7752	1	17	12367	11	1824
178	1	13	3489	4	101	7753	4	100	12377	5	57
207	3	34	3492	4	56	7755	1	10	12384	17	1075
241	1	10	3497	14	224	7756	3	33	12385	17	302
249	1	53	3527	1	10	7757	9	90	12389	14	1124
256	1	10	3536	1	23	7758	7	94	12390	1	21
268	1	11	3539	1	30	7761	1	20	12391	15	2487

Publisher	Detecting Test	Total Installs	Publisher	Detecting Test	Total Installs	Publisher	Detecting Test	Total Installs	Publisher	Detecting Test	Total Installs
272	1	14	3540	1	11	7763	5	76	12398	4	41
282	1	10	3543	1	53	7764	1	18	12400	7	126
322	13	207	3553	21	228	7773	3	33	12404	1	16
324	3	51	3560	1	23	7776	5	79	12408	1	13
330	1	13	3562	3	148	7778	1	12	12409	6	69
331	1	10	3565	1	33	7779	3	38	12412	12	137
336	1	15	3567	1	516	7780	76	992	12420	1	10
343	1	15	3568	1	2103	7803	1	22	12421	1	10
345	1	19	3569	4	1402	7809	1	12	12422	3	35
352	1	17	3570	5	164	7811	1	12	12430	1	20
360	1	22	3571	1	23	7812	38	385	12437	1	18
362	1	13	3572	1	337	7813	32	332	12442	1	10
365	1	10	3573	4	70	7818	17	179	12443	1	12
369	1	14	3574	5	67	7829	41	489	12444	1	24
371	1	12	3575	4	177	7830	16	247	12447	5	71
382	27	270	3577	1	45	7842	19	589	12450	15	174
383	1	19	3578	1	54	7851	17	297	12453	4	49
385	98	1003	3579	1	33	7855	25	325	12454	21	353
387	1	11	3580	1	105	7871	5	65	12456	3	183
405	1	59	3581	1	232	7872	1	26	12457	1	41
406	1	14	3582	1	96	7873	1	10	12463	4	49
407	1	12	3583	3	140	7948	9	123	12467	1	12
408	1	20	3584	3	32	7952	1	14	12468	12	135
409	1	15	3587	1	84	7957	4	70	12471	1	19
427	1	34	3588	3	686	7958	1	25	12502	1	17
441	1	10	3590	4	56	7961	3	42	12507	3	792
447	53	708	3591	1	309	7962	3	30	12513	1	44
498	34	678	3593	1	26	7963	1	14	12515	1	25
509	39	509	3594	1	215	7965	1	13	12522	1	19707
593	1	40	3595	1	43	7970	1	19	12525	8	365
603	1	12	3598	1	13	7973	4	248	12543	17	265
614	1	12	3599	1	12	7976	1	16	12570	15	162
623	1	17	3602	30	310	7981	9	101	12609	5	57
650	1	20	3614	32	335	7982	7	1950	12610	3	31
658	1	13	3615	26	269	7984	13	1843	12613	1	30
663	1	32	3633	5	74	7985	1	11	12620	4	57
665	1	21	3637	10	139	7986	10	286	12623	13	236
676	1	18	3638	5	339	7987	14	157	12624	1	121
678	1	13	3639	1	468	7988	7	76	12626	1	10

Publisher	Detecting Test	Total Installs	Publisher	Detecting Test	Total Installs	Publisher	Detecting Test	Total Installs	Publisher	Detecting Test	Total Installs
687	1	22	3641	1	129	7989	4	309	12627	1	72
691	4	41	3642	1	32	7993	3	44	12628	1	60
692	3	68	3651	79	833	7994	7	143	12629	1	10
693	1	10	3652	1	10	8010	1	37	12638	1	32
710	46	469	3654	1	99	8011	1	10	12679	6	597
718	4	68	3655	1	10	8015	1	11	12680	5	51
720	5	53	3657	85	865	8016	1	11	12693	1	10
721	8	237	3659	1	67	8017	3	47	12695	1	121
724	4	91	3667	4	41	8018	1	11	12697	3	50
725	9	182	3668	1	68	8037	18	185	12720	1	31
746	14	10740	3671	7	371	8117	1	10	12721	1	20
747	1	64	3703	1	21	8118	1	16	12722	1	12
748	1	31	3704	3	255	8119	1	10	12730	1	12
772	1	21	3705	1	283	8120	1	11	12742	9	115
822	1	32	3706	1	36	8121	1	17	12745	1	22
847	3	100	3707	4	51	8122	1	12	12746	3	50
848	1	19	3713	1	17	8124	1	12	12747	19	877
849	1	12	3717	4	167	8125	1	23	12748	1	11
850	1	10	3723	3	62	8187	18	248	12751	5	201
851	3	170	3724	1	13	8205	6	80	12758	1	10
852	1	33	3729	16	160	8220	17	217	12767	3	42
853	1	16	3731	24	252	8232	1	18	12771	1	17
854	1	16	3743	4	176	8233	1	11	12774	5	55
891	16	902	3744	3	41	8234	1	82	12776	6	70
895	1	23	3748	1	12	8235	1	15	12785	135	1353
902	1	34	3760	1	17	8237	1	284	12786	3	850
913	1	11	3761	3	50	8241	1	13	12787	1	396
947	1	45	3766	1	11	8314	1	11	12821	18	264
948	1	13	3830	7	79	8315	16	1170	12830	1	42
956	12	122	4093	1	20	8351	96	973	12834	4	52
957	8	84	4094	1	34	8355	30	373	12848	19	208
958	4	104	4142	1	24	8361	43	437	12849	3	40
960	1	24	4235	13	138	8362	27	626	12917	144	1785
965	1	49	4258	123	2723	8379	17	181	12968	1	15
966	1	16	4287	6	91	8380	24	417	12970	1	20
968	10	336	4333	1	49	8384	162	3191	12984	1	569
971	1	20	4334	1	85	8397	7	95	12986	1	24
991	6	462	4335	3	37	8411	43	474	12996	1	44
992	11	851	4338	1	10	8417	80	815	13001	3	45
993	3	167	4339	1	10	8437	22	1795	13002	20	493

Publisher	Detecting Test	Total Installs	Publisher	Detecting Test	Total Installs	Publisher	Detecting Test	Total Installs	Publisher	Detecting Test	Total Installs
994	3	44	4340	1	10	8482	42	433	13003	10	119
995	7	420	4341	21	304	8487	21	258	13020	20	213
996	1	131	4345	1	19	8493	9	3172	13062	1	85
998	4	127	4346	1	20	8495	110	1177	13063	6	93
1003	13	142	4351	1	25	8497	1	15	13064	9	118
1004	39	508	4385	40	400	8517	25	257	13065	4	58
1013	1	12	4410	1	43	8527	22	232	13067	1	148
1015	1	15	4411	1	30	8542	35	356	13069	1	30
1016	1	12	4413	1	73	8548	1	18	13071	4	73
1026	1	35	4415	1	32	8560	1	12	13072	1	232
1031	4	340	4420	1	24	8564	5	1364	13074	1	42
1035	10	376	4430	1	106	8576	12	1060	13075	3	1937
1037	7	129	4436	1	13	8615	1	35	13077	1	69
1038	1	53	4439	1	15	8632	13	184	13078	3	255
1042	1	17	4441	1	11	8635	68	684	13079	1	174
1043	1	17	4457	1	74	8638	1	41	13082	4	72
1048	1	16	4458	1	15	8649	1	14	13083	1	11
1051	3	74	4460	1	349	8650	1	36	13087	1	140
1053	1	18	4462	1	42	8651	1	41	13088	8	240
1054	3	43	4491	6	83	8653	1	510	13089	3	39
1061	1	13	4495	1	29	8654	3	834	13092	7	300
1062	1	12	4497	6	65	8655	3	741	13093	4	51
1063	1	12	4523	21	227	8656	1	49	13097	1	103
1065	1	20	4611	31	390	8658	1	92	13099	1	11
1066	1	13	4612	12	148	8659	3	122	13101	37	434
1067	1	21	4630	1	67	8660	1	153	13103	1	11
1075	1	21	4631	1	59	8661	1	75	13104	4	43
1076	1	10	4633	3	30	8662	1	15	13114	15	163
1080	1	12	4634	3	44	8663	4	56	13116	1	50
1090	1	27	4635	29	668	8664	1	29	13119	9	240
1093	16	327	4638	3	32	8665	3	148	13130	1	14
1094	3	38	4664	29	346	8668	3	148	13131	1	33
1095	1	28	4672	10	301	8669	4	41	13132	1	13
1096	1	14	4704	95	1096	8670	1	354	13134	1	17
1116	1	11	4804	1	24	8671	1	46	13136	1	19
1160	3	111	4812	3	411	8682	63	737	13138	3	252
1163	6	63	4898	44	559	8685	11	665	13140	21	325
1179	6	859	4900	3	194	8713	1	32	13141	3	94
1182	5	444	4919	6	67	8728	1	53	13142	13	634
1184	8	539	5003	5	57	8729	1	21	13143	4	109

Publisher	Detecting Test	Total Installs	Publisher	Detecting Test	Total Installs	Publisher	Detecting Test	Total Installs	Publisher	Detecting Test	Total Installs
1185	13	135	5019	6	104	8732	1	348	13144	6	151
1187	4	147	5025	20	428	8742	13	504	13145	10	450
1189	3	254	5027	1	25	8746	1	27	13146	1	10
1203	3	434	5029	1	11	8748	4	114	13150	1	45
1213	1	10	5030	3	101	8749	1	134	13151	7	76
1217	1	55	5037	1	51	8750	18	825	13152	1	50
1218	1	11	5042	19	198	8751	19	1312	13153	1	24
1221	1	19	5057	12	120	8761	16	224	13157	1	15
1231	1	16	5070	16	168	8767	12	593	13166	6	79
1232	1	11	5073	14	149	8769	19	274	13167	3	50
1242	1	10	5083	7	72	8782	9	149	13173	1	139
1288	14	152	5092	3	42	8809	4	55	13176	1	23
1289	6	69	5093	18	1525	8811	13	233	13180	1	130
1292	1	17	5094	1	40	8812	4	51	13188	1	14
1321	1	32	5095	1	20	8816	8	215	13190	13	1002
1340	5	287	5114	16	226	8817	4	49	13191	1	40
1341	10	557	5254	39	422	8825	7	72	13192	3	99
1342	1	15	5282	12	174	8863	3	36	13193	1	65
1343	1	22	5343	27	431	8864	1	28	13194	1	46
1345	4	428	5350	18	188	8865	6	131	13195	7	356
1346	3	71	5373	11	110	8870	4	58	13197	8	398
1348	1	11	5427	25	288	8875	22	257	13198	1	717
1361	5	199	5469	20	217	8895	1	43	13199	1	16
1363	3	51	5494	13	187	8896	1	12	13201	1	19
1366	1	13	5495	1	148	8897	1	38	13203	1	16
1368	1	32	5534	19	232	8901	1	15	13204	5	166
1370	1	18	5547	10	267	8906	1	20	13205	1	234
1373	3	43	5557	5	58	8936	55	566	13206	1	44
1374	9	1722	5610	19	260	8988	3	50	13208	1	86
1382	1	53	5621	1	38	8995	7	146	13214	4	98
1396	1	70	5623	10	116	9001	13	191	13220	3	284
1397	1	16	5625	47	993	9002	11	243	13224	3	49
1400	1	10	5648	43	760	9016	11	2073	13231	130	2851
1403	6	89	5650	10	187	9024	5	69	13234	1	178
1404	3	44	5651	1	12	9029	24	538	13241	1	214
1407	1	10	5654	1	10	9038	80	826	13242	1	319
1409	5	579	5655	1	29	9050	24	285	13245	1	57
1410	4	66	5661	1	17	9068	69	697	13246	3	157
1414	8	204	5662	1	78	9072	16	180	13247	1	18
1415	5	502	5724	20	247	9087	21	211	13248	1	486

Publisher	Detecting Test	Total Installs	Publisher	Detecting Test	Total Installs	Publisher	Detecting Test	Total Installs	Publisher	Detecting Test	Total Installs
1416	15	659	5729	3	86	9089	24	252	13249	1	20
1417	1	85	5775	28	1966	9099	9	92	13250	1	23
1419	25	553	5776	18	187	9102	6	65	13251	1	12
1421	12	129	5829	1	30	9103	18	187	13253	1	14
1422	20	609	5831	18	184	9109	1	14	13254	1	1489
1423	7	152	5877	4	46	9118	7	79	13255	1	411
1424	4	46	5901	3	36	9126	14	242	13256	1	53
1425	6	134	5904	1	43	9130	5	306	13258	377	3784
1426	14	1423	5931	26	321	9131	5	111	13260	1	12
1429	15	238	5934	1	72	9135	1	15	13262	47	1860
1431	9	1445	5954	3	64	9136	1	38	13264	1	17
1432	1	69	5960	3	65	9137	3	127	13267	1	18
1447	1	11	6036	18	273	9138	1	35	13268	5	1157
1458	10	214	6038	1	10	9139	1	14	13272	19	519
1510	8	111	6039	1	10	9142	7	109	13275	22	860
1513	19	449	6041	1	10	9144	3	53	13291	1	48
1518	6	88	6044	1	23	9148	1	23	13292	1	11
1527	5	86	6069	1	29	9172	21	255	13293	1	30
1531	6	232	6080	1	32	9280	10	472	13295	3	34
1540	1	28	6082	3	39	9286	1	11	13296	1	72
1549	13	295	6094	1	13	9288	1	22	13298	1	14
1550	7	277	6095	1	10	9294	3	117	13299	1	18
1555	1	15	6096	1	12	9314	1	16	13300	1	16
1557	3	60	6097	1	13	9315	1	88	13312	1	30
1560	3	415	6098	3	107	9316	1	26	13330	1	30
1562	5	53	6100	5	59	9319	3	78	13332	1	259
1601	1	12	6102	4	41	9322	5	73	13344	1	32
1605	1	16	6104	1	10	9324	1	19	13355	6	85
1606	1	12	6111	44	1581	9326	1	18	13375	18	2685
1617	1	12	6114	1	23	9328	4	142	13376	1	77
1647	18	263	6116	1	23	9336	14	199	13377	3	98
1654	27	567	6123	6	414	9343	18	1127	13379	3	56
1655	1	10	6124	1	11	9345	5	99	13404	1	38
1656	4	50	6125	5	51	9347	7	277	13405	1	23
1657	4	43	6128	1	10	9351	6	143	13411	1	10
1661	6	92	6137	1	14	9360	20	598	13412	3	367
1663	1	13	6141	7	88	9361	7	82	13415	5	98
1668	18	187	6173	15	2621	9362	11	145	13416	13	218
1671	103	1143	6174	1	24	9369	6	150	13417	1	10
1678	105	1157	6175	1	40	9370	15	164	13418	3	39

Publisher	Detecting Test	Total Installs	Publisher	Detecting Test	Total Installs	Publisher	Detecting Test	Total Installs	Publisher	Detecting Test	Total Installs
1693	1	15	6181	1	15	9371	7	82	13420	1	44
1697	1	25	6183	1	526	9372	8	85	13421	4	150
1698	1	38	6184	1	22	9373	11	125	13422	1	55
1699	5	60	6187	1	11	9403	1	44	13434	1	351
1704	3	35	6189	1	35	9408	1	15	13438	1	12
1719	1	22	6190	10	398	9415	1	10	13487	3	70
1720	1	22	6197	5	117	9417	1	19	13499	19	524
1721	1	18	6198	3	84	9418	1	21	13500	19	671
1722	1	15	6201	24	470	9420	1	16	13501	1	10
1723	1	11	6203	6	107	9439	1	10	13502	1	105
1747	12	132	6206	1	11	9440	1	13	13509	1	70
1760	130	1364	6208	4	51	9443	1	14	13510	1	23
1773	20	286	6214	1	10	9445	1	11	13513	1	29
1778	22	220	6216	4	115	9449	8	286	13514	1	48
1788	9	90	6218	5	303	9482	1	12	13516	1	22
1789	21	210	6222	1	20	9483	3	46	13518	1	42
1791	29	307	6225	4	53	9496	1	14	13520	1	112
1808	97	1465	6234	1	13	9535	1	12	13521	1	16
1813	37	1589	6239	1	10	9536	1	17	13522	1	53
1815	20	400	6240	1	19	9539	1	21	13527	1	10
1818	26	446	6241	1	10	9540	1	12	13545	1	16
1823	21	352	6351	3	238	9542	1	23	13548	7	90
1872	1	11	6352	1	194	9543	1	16	13549	1	84
1873	1	15	6354	1	77	9544	1	12	13582	27	317
1875	1	73	6358	8	104	9547	1	13	13620	1	46
1876	7	83	6359	1	313	9562	1	12	13621	1	23
1878	1	25	6366	1	11	9563	1	15	13622	1	12
1885	1	25	6370	1	32	9564	1	25	13637	1	118
1886	1	14	6371	5	92	9588	1	14	13641	4	41
1887	5	50	6376	1	13	9609	9	111	13642	1	128
1890	4	57	6377	1	20	9611	13	148	13644	1	10
1894	1	22	6379	1	13	9632	6	460	13649	1	19
1899	1	15	6382	5	54	9654	7	80	13652	4	204
1900	1	10	6383	15	774	9662	1	86	13654	1	88
1904	1	10	6384	9	376	9663	3	89	13657	3	40
1911	1	11	6387	1	11	9678	1	14	13663	1	10
1919	20	293	6388	3	44	9759	1	18	13667	1	14
1928	1	39	6389	1	21	9770	1	11	13682	1	14
1934	1	27	6390	1	29	9792	1	52	13684	1	11
1935	1	11	6391	1	30	9807	1	13	13692	1	15

Publisher	Detecting Test	Total Installs	Publisher	Detecting Test	Total Installs	Publisher	Detecting Test	Total Installs	Publisher	Detecting Test	Total Installs
1938	8	111	6392	1	26	9863	16	278	13693	1	24
1942	1	69	6393	1	31	9869	10	176	13696	1	72
1944	1	95	6394	24	323	9937	9	116	13697	1	53
1945	5	196	6396	15	191	9955	4	177	13701	5	88
1947	1	25	6397	1	213	9957	1	39	13737	337	5266
1951	1	20	6401	1	12	9959	4	105	13765	168	2218
1952	13	189	6406	16	382	9960	1	20	13823	401	9743
1956	1	17	6407	3	1237	9963	14	166	13867	6	60
1958	1	84	6409	16	187	9964	1	12	13871	120	1233
1961	1	65	6475	1	10	9966	1	60	13876	366	4162
1963	1	31	6476	1	10	9999	18	200	13879	15	196
1965	1	29	6482	1	17	10006	1	18	13910	1	20
1966	4	1810	6548	1	37	10009	9	91	13912	6	60
1971	5	216	6562	1	26	10010	7	199	13921	1	13
1972	1	32	6602	1	45	10011	1	27	13922	1	14
1973	7	362	6646	13	136	10082	1	15	13923	1	34
1975	4	74	6657	12	190	10084	6	135	13924	1	11
1976	7	160	6658	10	180	10086	1	35	13925	1	10
1977	1	69	6662	6	82	10087	1	10	13926	1	23
1978	1	13	6669	9	181	10088	1	34	13927	1	21
1979	1	29	6676	326	24253	10118	8	88	13928	3	581
1980	1	25	6677	5	64	10125	1	11	13931	1	47
1981	3	120	6683	1	17	10140	1	35	13934	1	13
1982	3	65	6684	1	52	10180	3	46	13935	1	10
1983	1	83	6687	5	445	10188	1	62	13936	4	52
1985	1	15	6780	9	686	10209	3	175	13938	1	31
1988	10	409	6783	8	1816	10211	1	11	13942	3	58
1990	4	53	6788	21	445	10212	1	24	13944	1	11
1991	1	11	6803	25	330	10222	9	110	13947	3	61
1992	5	176	6807	16	280	10227	1	12	13948	1	244
1993	5	91	6809	20	218	10228	7	97	13949	1	107
1994	1	15	6811	3	441	10245	24	331	13998	1965	19658
1997	1	45	6817	36	411	10281	1	34	14079	5	1116
2085	3	119	6818	4	524	10430	19	190	14096	4	2045
2086	1	25	6866	3	249	10442	1	13	14125	3	86
2095	1	12	6871	5	52	10461	1	14	14137	20	381
2155	1	41	6872	1	11	10493	5	54	14150	12	198
2165	22	233	6876	7	319	10498	1	21	14155	108	5763
2168	1	13	6877	1	22	10531	1	164	14163	8	82
2173	26	277	6878	1	14	10553	3	97	14168	6	79

Publisher	Detecting Test	Total Installs	Publisher	Detecting Test	Total Installs	Publisher	Detecting Test	Total Installs	Publisher	Detecting Test	Total Installs
2176	15	159	6940	11	112	10566	3	77	14174	8	145
2180	15	161	6943	7	81	10573	1	16	14175	4	51
2198	8	393	6960	1	11	10606	44	470	14184	6	60
2202	3	31	6963	1	59	10616	22	225	14195	9	402
2203	3	121	6969	7	128	10617	147	1477	14198	1	85
2210	110	1108	7004	1	29	10636	5	53	14209	1	77
2212	1	181	7018	1	16	10651	1	10	14263	80	1327
2220	26	1402	7022	1	15	10680	1	94	14267	1	55
2221	5	309	7023	1	22	10681	1	26	14272	1	55
2222	1	48	7024	1	19	10682	1	86	14278	5	230
2228	3	41	7025	1	30	10683	1	12	14279	31	1980
2230	3	106	7026	1	19	10684	1	14	14280	11	271
2236	4	64	7031	1	14	10685	1	79	14338	6	72
2240	1	43	7032	1	14	10686	1	10	14389	3	313
2241	4	166	7071	14	433	10722	97	2287	14398	3	292
2260	1	11	7085	1	35	10758	1	80	14448	8	106
2262	1	10	7092	1	65	10774	20	1038	14453	52	542
2266	1	10	7102	1	20	10788	3	98	14463	29	438
2268	3	52	7104	1	16	10789	1	27	14475	9	103
2269	1	57	7112	1	40	10891	279	2832	14489	8	85
2275	13	328	7122	241	2891	10947	11	198	14491	1	220
2276	4	56	7129	17	218	11075	9	95	14526	3	36
2280	4	827	7131	1	11	11170	104	1179	14634	48	1013
2285	5	155	7165	1	47	11175	22	231	14637	54	1549
2286	5	279	7170	320	5158	11181	12	140	14650	51	801
2287	5	59	7211	45	1025	11201	1	59	14655	10	138
2288	3	66	7217	17	442	11202	1	29	14660	1	22
2295	11	276	7237	8	87	11230	1	63	14697	44	536
2296	1	29	7238	12	143	11233	1	18	14698	15	266
2298	1	18	7248	21	222	11319	7	157	14699	4	336
2299	3	118	7257	3	40	11324	8	163	14700	26	968
2301	9	247	7260	6	69	11365	3	60	14702	14	170
2307	1	27	7267	91	1653	11366	17	367	14706	3	1138
2308	1	18	7271	12	185	11368	4	63	14707	1	100
2316	5	1222	7272	66	5238	11370	5	73	14711	16	167
2317	4	414	7286	121	1480	11373	5	81	14713	1	132
2321	1	514	7290	1	157	11374	12	176	14715	1	448
2328	3	47	7293	4	113	11377	4	42	14716	15	226
2329	3	210	7308	22	367	11378	22	322	14718	1	11
2341	1	79	7309	1	38	11380	6	218	14737	5	217

Publisher	Detecting Test	Total Installs	Publisher	Detecting Test	Total Installs	Publisher	Detecting Test	Total Installs	Publisher	Detecting Test	Total Installs
2346	1	97	7329	1	4584	11382	3	414	14759	18	247
2347	1	19	7331	1	31	11386	1	39	14772	1	12
2350	1	16	7333	9	99	11387	10	300	14780	24	1045
2354	5	87	7334	762	8796	11424	14	265	14862	266	8217
2357	1	32	7337	78	994	11461	1	82	14908	1	42
2360	1	16	7338	208	25625	11472	13	207	14918	3	251
2384	59	629	7356	87	1019	11484	21	246	14919	1	42
2417	6	74	7357	37	1637	11504	15	183	14921	1	29
2434	1	18	7379	3	37	11508	36	497	14960	3	58
2435	1	33	7394	657	8424	11544	5	60	15044	47	571
2436	3	38	7396	42	676	11548	29	312	15200	13	176
2442	1	25	7411	14	628	11552	12	408	15202	1	15
2463	1	22	7439	32	401	11641	7	105	15230	26	271
2470	6	64	7455	22	272	11660	21	220	15231	21	266
2475	1	11	7460	29	334	11676	15	176	15233	7	72
2481	7	617	7467	195	1964	11680	44	529	15253	1	26
2482	9	820	7472	6	61	11689	4	49	15263	17	3863
2483	9	524	7530	1	54	11755	1	15			
2562	38	7323	7532	1	47	11761	1	10			
2588	47	509	7541	5	66	11781	1	12			