Click Fraud Detection Approaches to analyze the Ad Clicks Performed by Malicious Code

Click Fraud Detection Approaches to analyze the Ad Clicks Performed by Malicious Code

Mahesh Bathula[1], Rama Chaithanya Tanguturi[2], Srinivasa Rao Madala[3]
[1]M.Tech., Scholar, [2]Professor, [3]Professor & HOD
Department of Computer Science and Engineering
PACE Institute of Technology and Sciences

Abstract. Mobile PR is an important component of the mobile app ecosystem. A major threat to this ecosystem's long-term health is click fraud, which involves clicking on ads while infected with malware or using an automated bot to do it for you. The methods used to identify click fraud now focus on looking at server requests. Although these methods have the potential to produce huge numbers of false negatives, they may easily be avoided if clicks are hidden behind proxies or distributed globally. AdSherlock is a customer-side (inside the app) efficient and deployable click fraud detection system for mobile applications that we provide in this work. AdSherlock separates the computationally expensive click request identification procedures into an offline and online approach. AdSherlock uses URL (Uniform Resource Locator) tokenization in the Offline phase to create accurate and probabilistic patterns. These models are used to identify click requests online, and an ad request tree model is used to detect click fraud after that. In order to develop and evaluate the AdSherlock prototype, we utilise actual applications. It injects the online detector directly into an executable software package using binary instrumentation technology (BIT). The findings show that AdSherlock outperforms current state-of-the-art methods for detecting click fraud with little false positives. Advertisement requests identification, mobile advertising fraud detection are some of the keywords used in this article.

1.Introduction

The ecosystem of mobile applications relies heavily on mobile advertising and mobile public relations. In 2020, mobile advertising expenditure is expected to exceed $247.4 billion globally, according to recent research[1]. Generally, app developers integrate ad libraries from third-party ad providers like AdMob[2] for app advertising. When mobile users use this app, the integrated ad library gets ad content from the network and displays it to them as ads. If someone clicks on an ad that uses PPC (Pay-Per-Click)[2-6], the developer and ad supplier both get money from the advertiser. Click fraud[7-10] poses a significant threat to the ecosystem's long-term sustainability since it involves rogue programmes clicking on ads in a programmatic or automated bot fashion (i.e. touch events on mobile devices).

There are a lot of different ways to commit click fraud, however there are generally two types: fraud For false ad clicks, malicious code is included in the app; bot-driven frauds utilise bot programmes to click advertisements automatically, i.e. a fraudulent app. The MAdFraud[5] recently

conducted a research to quantify in-app ad fraud by measuring a wide range of ad fraud in real-world applications. Ad requests are generated when the app is running in the background, according to MAdFraud's analysis of 130K Android apps. Using an automated ClickDroid[4] software, a new study on bot-driven click fraud is testing eight well-known advertising networks by committing real click fraud against them. According to the findings [11,12], six of the eight major PR networks are vulnerable to cyberattacks. In order to detect click fraud in mobile apps, a threshold-based detection method is used on the server side. Ad servers may suspect fraud if they get a high number of clicks from the same device ID (such as an IP address) in a short period of time.

Because clicks behind proxies or scattered globally may be easily circumvented when using this simple method, it's possible to get a lot of false negatives. In the literature, there are even more sophisticated methods[6],[7] for detecting click fraud on the server side. Click fraud may still be an issue despite server-side techniques having more accuracy. For example, in the current mobile ad fraud competition [6], the three best methods only obtain an accuracy of 46.15 percent to 51.55 percent utilising various machine learning technologies. As a result of the shortcomings of server-side techniques, a new problem has emerged: how should we approach clients? On the contrary, server-side methods make it more difficult to tell if a user has interacted at all. However, because the developer receives compensation for fraudulent ad clicks, the attacker may be the developers themselves. We can't rely on developers to coordinate the click fraud detection SDK, for example, by developing a customer-side approach, because of the inherent conflict of interest. As a result, we focused on identifying click fraud in mobile applications using a customer-centric approach rather than working with developers.

There are two major issues with this system's design. To begin with, the computer, memory, and energy resources of a mobile client are constrained. As a result, the proposed method must carry out the whole fraud detection process efficiently and without imposing a significant overhead. As a result, it seems that current server-side learning methods are inadequate for detecting click fraud. Second, rather than in a controlled environment devoted to fraud detection, real situations should be used to identify click fraud. By using a controlled environment, MAdFraud[13] may examine the advertising default behaviour of several different apps at once; only one application is active at a time, with all HTTP requests being logged for further study. However, in our case, fraud detection should take place in real-world situations without the use of a third party app. We provide AdSherlock in this post, an effective and deployable method for mobile app click fraud detection from the customer's side. Because it's a client-side solution, AdSherlock isn't compatible with existing server-side techniques.

Appropriate for app stores, AdSherlock is designed to ensure a healthy mobile app ecosystem. In-service fraud as well as fraud-led may be combated because to AdSherlock's high degree of accuracy. Keep in mind that AdSherlock may be used to detect fraud in-app by anybody. Advertisers may utilise AdSherlock, for example, to see whether the apps that use their libraries are genuine. A lightweight online fraud detector and an accurate offline model extractor are used by AdSherlock to accomplish these goals. AdSherlock is divided into two stages. The offline pattern extractor runs each app automatically in the first phase and collects traffic patterns for efficient ad request identification. This is the second step. After tokenizing network requests, AdSherlock generates accurate and probabilistic patterns for robust matching.

The offline pattern extractor in AdSherlock allows it to do computationally and I/O-intensive design-generation activities without degrading online fraud detection operations. The online fraud

detector and the patterns created in the first phase are both incorporated into the programme in the second phase and used in actual user scenarios. Using AdSherlock's request tree structure, the programme is able to accurately and rapidly identify click requests. It's possible to obtain the finely grained user input events needed to identify fraud since the online fraud detector is included within the app, making it available to you. App shops should use AdSherlock, which is a tool from the AdSherlock team. It's possible that the app store may use AdSherlock to analyse an app before it's published for download so that it can be used as a fraud detector online in the click fraud detection app. There is no developer input allowed by AdSherlock and just application binaries are required.

For the most part, AdSherlock is made up of an extractor and a fraud detector. Before it can begin collecting network traffic, it must accept the programme as input and execute it in the background. The pattern extractor for use when you're not connected to the internet. It then separates the ad traffic from the non-ad traffic by classifying the patterns. Once the traffic patterns have been gathered, the online fraud detector may be built. The online fraud detector relies on network traffic monitoring, ad request identification, and click fraud detection. In the end, AdSherlock implements the online fraud detection in an app binary, which is then made available via the app store's distribution channels. Figure 1 depicts AdSherlock's core components. Each app is loaded into the offline pattern extractor once it has been published to the app store. Ads and non-ad traffic patterns are generated automatically by this extractor. The Online Fraud Detector and these patterns are included into the software. Online fraud detectors utilise patterns generated offline to identify ad requests as soon as the software is launched on the end user's device. Creating an ad request tree can help identify the click request. Anomalies in click requests are discovered via an examination of user input events. This is a quick and easy method. We identify a bogus request for clicks as one that is not accompanied by any real input events.

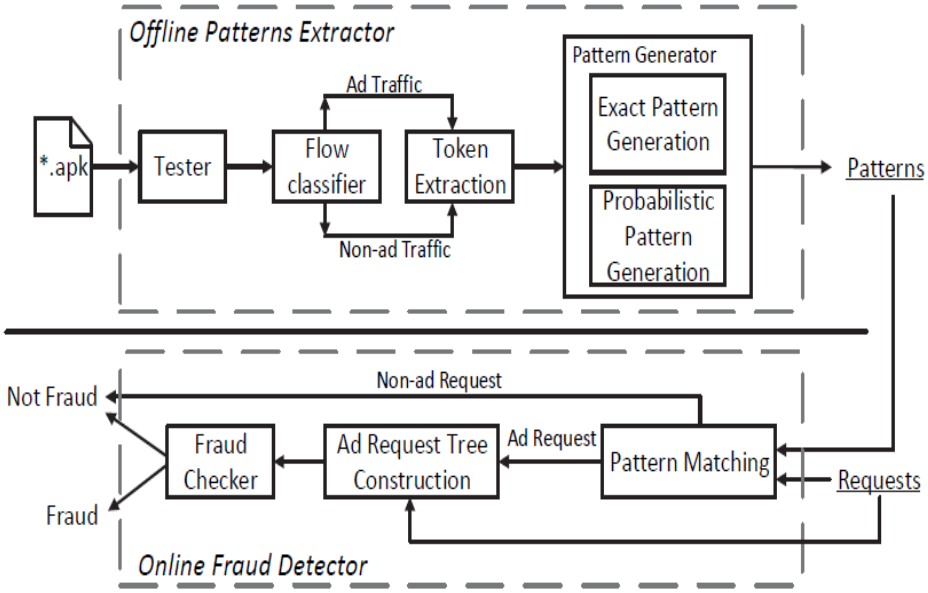


Fig 1. Overview of AdSherlock.

2.Related Works

Bots-driven click fraud in online advertising is the primary focus of click fraud detection research. In most cases, these methods are implemented on the server side and focus on analysing network data and identifying click-fraud behaviour. [8] and [9] use the IP addresses of clients and their cookie IDs

to build a list of customers who have deviated from standard ad-traffic behaviour. SBotMiner[10] searches for anomalies in query distribution to identify search engine bots. There is a risk that sophisticated IP address and traffic bots may spoof your IP address if you use these services. The AdSherlock customer-side approach utilises a terminal device's hard-to-override click event characteristic, unlike those other methods. These server-side methods must also gather enough ad traffic for analysis without the need of AdSherlock software. As soon as a fraudulent click is detected on the client side, AdSherlock may stop it in its tracks. Detecting duplicate clicks, which occur when a publisher clicks on the same ad again, is the subject of other initiatives like [11] and [12]. As a supplement to AdSherlock, these server-side methods may detect real human click fraud. There is a close link between FcFraud [13] and our work since it's the newest attempt in online advertising to identify click fraud. Advertisement clicks are recognised, and whether or not they are accompanied by real mouse actions is determined by the tool. Andriod applications will have a long-term burden of dealing with a large number of HTTP queries in order to categorise ad requests. AdSherlock, on the other hand, aims to detect click fraud in mobile apps.

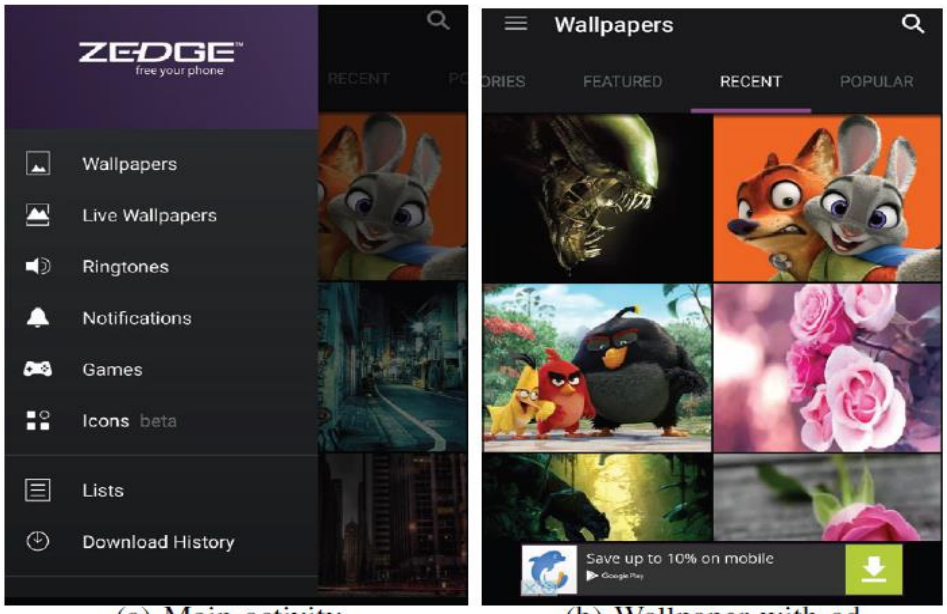
Mobile ad fraud has been the subject of many recent initiatives. MadFraud[15-18] investigates input fraud in ad fraud detection using Android emulators and apps that track deviant behaviour. Display fraud applications such as small advertising, intrusive advertisements, and so on are examined by DECAF [14]. Applications are evaluated using DECAF. In a controlled environment, they're nonetheless well-researched and tough to detect click fraud bots powered. AdSherlock is implemented in a new production environment and online fraud is detected by clicking.

Click-through fraud was the subject of yet another recent paper[25-28]. A ClickDroid [21] automated tool for simulating and detecting fraud is created by distinguishing between real-world and programmed tactile events. The Android kernel must be modified to remove touch events produced by applications. AdSherlock is a generalised method for combating both in-service click fraud and proactive in-service click frauds that requires no modification to the Android kernel. There is also a hardware-assisted technique for detecting mobile publicity fraud[19]. To demonstrate an unforgivable click and verified display, AdAttester[20] uses the TruseZone ARM. AdSherlock, in contrast to AdAttester, does not require any additional hardware support.

3.Proposed System

Ad traffic and non-ad traffic make up the majority of network requests. For each application, we aim to eliminate ad and non-ad traffic. These recovered patterns are made up of sets of substrates found in network traffic that differentiate between asynchronous and synchronous communication. Here, you'll learn about how traffic patterns may be automatically extracted, as well as some of the challenges that come with it. After that, we'll go into detail about how the pattern was made. Extraction of patterns relies on setting up invariant portions of network requests. A highly rated wallpaper and ringtone downloader, Zedge[18] is examined to better understand the driving forces and issues. While using Zedge, network requests are generated for various network activities, such as loading advertisements, previewing wallpapers, and downloading them. The following are examples of user interactions: Zedge: Once the app has been launched, the user clicks on a topic of interest, such as wallpapers, and a list of available wallpapers and an ad at the bottom of the page is displayed (see Fig. 2(b)). A wallpaper can then be downloaded, or an ad of interest can be selected for further consideration and action. Network characteristics with and without ad traffic are shown in Fig. 3 in a similar way. To make HTTP requests easier to comprehend, just the GET method and URI are given here. These concepts are likely to be used elsewhere as well. HTTP header elements like Host, URL path, and URL query include a lot of invariant data. This holds true for HTTPS headers as well. Online fraud detection is done in-app since we think HTTPS traffic may be intercepted before encryption. For the sake of simplicity, we'll just look at HTTP traffic in this research. HTTPS is not used by more than 70% of applications, according to the Dai et al. To ensure that ad material is accessible, the ad provider typically provides CDN services. CDN traffic may also be utilised as an ad traffic. Extraction

of traffic patterns is still based on the same concept. It is our primary goal to create patterns with high quality patterns that have minimal levels of misleading advertising positives and negative advertising consequences. Three practical issues must be addressed for high-quality model creation to take place: There are many types of ad requests that may be handled with ease. Due to the fact that an app may include several ad libraries, a variety of ad requests may be produced. The various types of ad requests have significant differences. For example, as shown in Fig. 3, Zedge requests advertisements on MoPub [20] and Admob [2]. (b). When it comes to ad traffic, the longest invariant substratum is "GET," which renders non-ad traffic as 100 percent false. Because of this, a single ad traffic pattern has a skewed balance of good and negative impacts. As an alternative, we develop a large number of patterns, each of which corresponds to a subset of a category's request. Low positive and negative patterns may be found in a variety of patterns.



(a) Main activity (b) Wallpaper with ad

Fig. 2: The screen shots of Zedge.

```
GET /dl/wallpaper/9f1f4c3bc6930c04aa270c4160f0dbe8/
judy_and_nick.jpg?ref=android&type=mc&attachment=1 HTTP/1.1
Host: fsa.zedge.net

GET /dl/ringtone/222e5f3c1b7451b2ba8cf45dd3391af3/
lollipop.mp3?ref=android&type=mc&attachment=1 HTTP/1.1
Host: fsa.zedge.net

GET /dl/notification_sound/74c7e5d210dcaef6d1a3caa1bf66320b/
blackberry.mp3?ref=android&type=mc&attachment=1 HTTP/1.1
Host: fsb.zedge.net
```

(a) Non-ad traffics

```
GET /m/ad?v=6&id=3ee536462fd649aba0abd161bfadf256&...&fail=1
HTTP/1.1
Host: ads.mopub.com

GET /m/imp?appid=&cid=105ba158d8874b00868c3e48d06dbbe6&city=
Hangzhou&...&video_type= HTTP/1.1
Host: ads.mopub.com

GET /mads/gma?request_id=79835a07-ec6c-47c9-bc5b-5e79b7ba889e
&carrier=310260&...&url=1499 HTTP/1.1
Host: googleads.g.doubleclick.net
```

(b) Ad traffics

Fig. 3: The network traffics of Zedge.

4.Results And Discussion

AdSherlock has been prototyped and implemented. Using Ubuntu 14.04's four core 3.30 GHz CPU and 12GB RAM, the Python Offline Model Extractor may be used. Using a Nexus 5 smartphone with a 2.26GHz quad-core processor and 2GB of RAM, a simple Android app is developed to target Android API level 19. The detector is a simple online fraud scanner. The online fraud detector is inserted into the archive of the application via binary gadgets. During operation, it intercepts network data and logs user input events in the buffer. To detect ad requests, the network traffic is sent into the appropriate pattern section. The fraud checker is used to detect touchscreen input events, i.e. motion events, that are fraudulently performed by users.

Google Play has 18,606 apps in its library as of November 2017. Static studies are then performed on those apps in the ten app categories that were chosen because they have embedded ad libraries. Ads are present in 61.3% of all apps, with the majority coming from popular categories such as Entertainment, Customization, Music & Audio, and Casual, among others. After that, we choose 1750 free apps that don't need logins and send HTTP requests to at least one well-known ad provider. We put it through its paces in our Tester and record the network traffic it generates. With the help of the most common ad libraries, we build a collection of fundamental truth facts that may be used to identify ad requests. These ad sites' ad requests will continue to be recognised by us. A total of 230,626 traffic events have generated 16,751 ad applications.

Fig. 4 depicts the outcome of fraud detection in the botdriven scenario. It is the identification of ad requests that determines the accuracy of fraud detection in this scenario The strong recall of AdSherlock and MadFraudS may be shown in Figure 4(a). Figure 4(b) and (c) show that they are more accurate and have a higher F1– than MadFraudS. AdSherlock has a better handle on things. Fraud detection is shown in Fig. 5 in the application scenario. There are more false positives in Fig. 6(a) than correct patterns. Probabilistic patterns have a success rate of above 98 percent in all applications. Due to probabilistic patterns being more traffic resistant and having better impacts, this is true. Figure 6(b) shows that the exact pattern has a lower false positive rate than the general pattern. Both patterns have a low false positive rate (less than 0.4 percent) across all applications. In probabilistic patterns, false positives outnumber accurate patterns eight times out of ten. Because probabilistic patterns are more flexible, it's a reasonable assumption to make. Figure 6 shows probabilistic models that have less accuracy

(c).

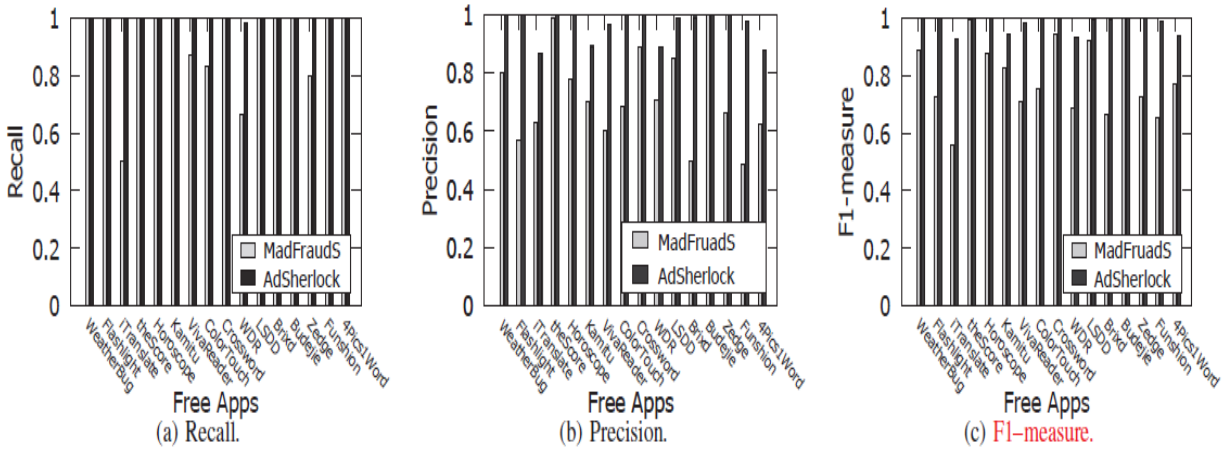


Fig. 4: Performance of click fraud detection in bot-driven fraudulent scenario.

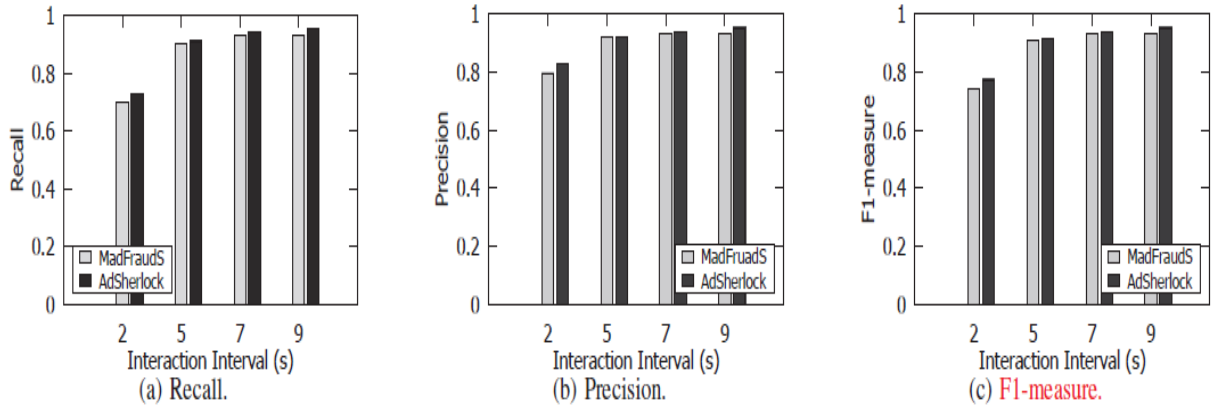


Fig. 5: Performance of click fraud detection in in-app fraudulent scenario.

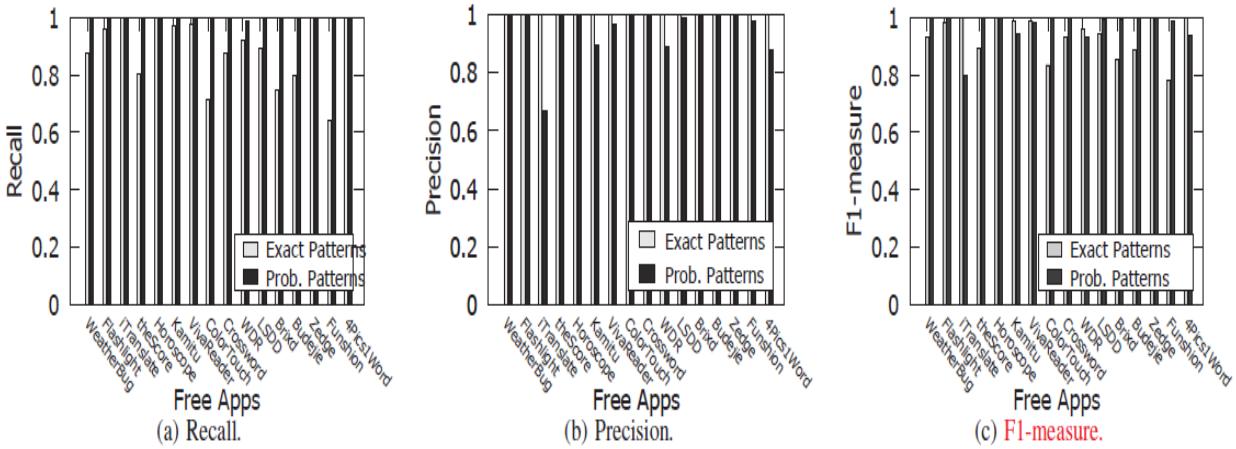


Fig. 6: Performance of exact patterns and probabilistic patterns generated by AdSherlock.

5.Future Scope And Conclusion

AdSherlock is an effective and deployable technique for detecting client clicks. As a client-side approach, AdSherlock differs from existing server-side techniques. An offline and an online method are used to split the heavy computation required for the identification of click requests. AdSherlock's offline method uses url tokenization to generate both precise and probabilistic patterns for ad designs. In combination with an ad request tree model for click fraud detection, these patterns are utilised all through the online process to identify click requests and click fraud patterns. According to the results, AdSherlock is effective at detecting click fraud even at high levels of click volume. Adsherlock-escape attacks will be investigated by combining static analysis with road testing in the future, which will help to improve ad request recognition accuracy.

References

[1] "Mobile advertising spending worldwide." [Online]. Available: https://www.statista.com/statistics/280640/mobile-advertisingspending- worldwide/
[2] "Google admob." [Online]. Available: https://apps.admob.com/
[3] M. Mahdian and K. Tomak, "Pay-per-action model for online advertising," in *Proc. of ACM ADKDD*, 2007.
[4] G. Cho, J. Cho, Y. Song, and H. Kim, "An empirical study of click fraud in mobile advertising networks," in *Proc. of ACM ARES*, 2015.
[5] J. Crussell, R. Stevens, and H. Chen, "Madfraud: Investigating ad fraud in android applications," in *Proc. of ACM MobySys*, 2014.
[6] R. Oentaryo, E.-P. Lim, M. Finegold, D. Lo, F. Zhu, C. Phua, E.-Y. Cheu, G.-E. Yap, K. Sim, M. N. Nguyen, K. Perera, B. Neupane, M. Faisal, Z. Aung, W. L. Woon, W. Chen, D. Patel, and D. Berrar, "Detecting click fraud in online advertising: A data mining approach," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 99–140, 2014.

[7] B. Kitts, Y. J. Zhang, G. Wu, W. Brandi, J. Beasley, K. Morrill, J. Ettedgui, S. Siddhartha, H. Yuan, F. Gao, P. Azo, and R. Mahato, *Click Fraud Detection: Adversarial Pattern Recognition over 5 Years at Microsoft*. Cham: Springer International Publishing, 2015, pp. 181–201.
[8] A. Metwally, D. Agrawal, and A. El Abbadi, "Detectives: detecting coalition hit inflation attacks in advertising networks streams," in *Proc. of ACM WWW*, 2007.
[9] A. Metwally, D. Agrawal, A. El Abbad, and Q. Zheng, "On hit inflation techniques and detection in streams of web advertising networks," in *Proc. of IEEE ICDCS*, 2007.
[10] F. Yu, Y. Xie, and Q. Ke, "Sbotminer: large scale search bot detection," in *Proc. of ACM WSDM*, 2010.
[11] L. Zhang and Y. Guan, "Detecting click fraud in pay-per-click streams of online advertising networks," in *Proc. of IEEE ICDCS*, 2008.
[12] A. Metwally, D. Agrawal, and A. El Abbadi, "Duplicate detection in click streams," in *Proc. of ACM WWW*, 2005.
[13] M. S. Iqbal, M. Zulkernine, F. Jaafar, and Y. Gu, "Fcfraud: Fighting click-fraud from the user side," in *Proc. of IEEE HASE*, 2016.
[14] B. Liu, S. Nath, R. Govindan, and J. Liu, "Decaf: detecting and characterizing ad fraud in mobile apps," in *Proc. of USENIX NSDI*, 2014.
[15] G. Cho, J. Cho, Y. Song, D. Choi, and H. Kim, "Combating online fraud attacks in mobile-based advertising," *EURASIP Journal on Information Security*, vol. 2016, no. 1, p. 1, 2016.
[16] W. Li, H. Li, H. Chen, and Y. Xia, "Adattester: Secure online mobile advertisement attestation using trustzone," in *Proc. of ACM MobySys*, 2015.
[17]"Monkeyrunner." [Online]. Available: http://developer.android.com/studio/test/monkeyrunner/index.html
[18] "Zedge." [Online]. Available: https://play.google.com/store/apps/
[19] S. Dai, A. Tongaonkar, X. Wang, A. Nucci, and D. Song, "Networkprofiler: Towards automatic fingerprinting of android apps," in *Proc. of IEEE INFOCOM*, 2013.
[20] "Mopub." [Online]. Available: https://www.mopub.com/
[21] J. Newsome, B. Karp, and D. Song, "Polygraph: Automatically generating signatures for polymorphic worms," in *Proc. of IEEE S&P*, 2005.
[22] "Android montionevent." [Online]. Available: https://developer.android.com/reference/android/view/MotionEvent.html
[23] X. Jin, P. Huang, T. Xu, and Y. Zhou, "Nchecker: saving mobile app developers from network disruptions," in *Proc. of ACM EuroSys*, 2016.
[24] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *Proc. of ACM ICML*, 2006.
[25] T. Saito and M. Rehmsmeier, "The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets," *PloS one*, vol. 10, no. 3, p. e0118432, 2015.
[26] Madala, S. R., Rajavarman, V. N., & Vivek, T. V. S. (2018). Analysis of Different Pattern Evaluation Procedures for Big Data Visualization in Data Analysis. In Data Engineering and Intelligent Computing (pp. 453-461). Springer, Singapore.
[27] Madala, S. R., & Rajavarman, V. N. (2018). Efficient Outline Computation for Multi View Data Visualization on Big Data. International Journal of Pure and Applied Mathematics, 119(7), 745-755.
[28] Vivek, T. V. S., Rajavarman, V. N., & Madala, S. R. (2020). Advanced graphical-based security approach to handle hard AI problems based on visual security. International Journal of Intelligent Enterprise, 7(1-3), 250-266.