Who Moved My Cheese: Towards Automatic and Fine-grained Classification and Modeling Ad Network

Jiafa Liu*, Junyi Liu*, Huaxin Li*, Haojin Zhu*, Na Ruan*
* Department of Computer Science and Engineering
Shanghai Jiaotong University
Email: {10421381150021, liujunyi, lihuaxin003, zhu-hj, naruan}@sjtu.edu.cn
Di Ma†
† University of Michigan-Dearborn
Email: dmadma@umich.edu

***Abstract*—The mobile advertisement (ad) network is gaining an increasing interest due to the high popularity of smart phones. Previous researches on the security issues of ad network primarily focus on the privacy, permission and malware detection while less attention has been paid to the traffic consumption issue incurred by ad network. Though it is well known that ad network plays an important role in network consumption, it represents a great challenge of giving a fine-grained classification of ad networks. Inspired by this, different from any previous researches, in this study, we take the initial step towards modeling the network consumption of Ad network in Android. We develop an automatic ad analysis platform to quantify the ad network traffic consumed by android applications (app). To achieve a fine-grained quantification, we combine two sources of network traffic. On one hand, we modify the android webview and log system in system level to capture network traffic accurately. On the other hand, we capture network traffic in router level to collect detailed information of traffic packets, such as packet size and URI. We have evaluated the developed system in terms of normal apps, repacked apps and malicious apps based on the real-world dataset, which is comprised of 93 Android apps. We find out that ad traffic takes major percentage of the whole network traffic caused by Android app. We have also studied the ad library mechanism for 10 popular ad libraries. We found ads from some ad libraries use much more network traffic because they have to be fetched from remote ad libraries each time they are shown to users while other ad libraries allow apps to store ads locally.**

I. Introduction

Along with the popularity of smartphone as well as mobile devices, we have witnessed a tremendous increase in mobile data usage. According to the latest report from CISCO [1], smartphones represented only 43 percent of total global handsets in use in 2015, but generated 97 percent of total global handset traffic. Average smartphone usage grew 43% in 2015. The typical smartphone generated 41 times more mobile data traffic (929 MB per month) than the typical basic-feather cell phone (which generated only 23 MB per month of mobile data traffic). These statistics clearly reflect the growing popularity of smartphone usage amongst mobile data users, which will further boost the growth of mobile data traffic.

As part of the mobile eco-system, the app developers, largely motivated by financial incentives, submit their apps to third-party app store for users to access. However, the majority of apps are free to download. To compensate for their work, many app developers incorporate an advertisement library (also known as an ad library) in their apps. The ad library fetches ads from the ad network's servers to display. After users have viewed the ads, the app developer will get paid correspondingly. During the whole process, mobile ads contribute a significant overhead of mobile traffic. In addition to ad network, another important factor that contributes to the extra mobile traffic is the malware. The existing researches pointed out that about 75% of malware involves network activities and generates some HTTP traffic [14].

Existing work focuses on analyzing smartphone data traffic [17]. However, they fail to provide a fine-grained and systematic evaluation on various traffic overhead incurred by smart phones. It is highly expected for an in-depth analysis on the traffic overhead incurred by the smartphone apps.

In this paper, we aim to give a systematic study on characterizing and quantifying the overhead traffic incurred by smartphone apps. The contributions of this work are summarized as follows:

- We develop an automatic system which aims to automatically capture the traffic, classify the ad traffic from the service traffic in terms of hosts, aggregate the overall traffic consumptions. The developed system is expected to cope with various kinds of apps, including repacked apps and even malicious ones.
- We investigate 10 popular ad libraries and perform a detailed evaluation on their impacts on the mobile traffic. At the same time, our platform can calculate the total network traffic caused by each ad library.
- In the evaluation, we compare the triggered traffic overhead of original apps, repacked apps and malicious apps. The dataset covers 1000 normal apps from Google Play, 320 repacked apps from PANGU research lab [11] and malicious apps from Android Malware Genome Project

978-1-5090-1328-9/16/$31.00 ©2016 IEEE

[12]. Our study shows it is surprising that the repacked apps generally consume the most ad network traffic and malicious apps consume the least ad network traffic.

The remain of this paper is organized as follows. In section II, we introduce related work of this paper. Section III shows the research goals of our paper. Section IV describes our system design about how to extract ad traffic. Section V discusses the implementation of our system. Section VI provides detailed results of various kinds of experiments about ad traffic. Finally, we come to a conclusion in section VII.

II. Related Work

There are two previous research studies on the composition and consumption of smartphone network traffic [6] [13]. These two papers respectively captured the network traffic of android and iPhone users and analyzed the composition. However, none of them considered ad traffic. Another work focused on the overhead traffic caused by free apps [17], but they were unable to quantify the traffic automatically and precisely. In [9], it analyzed the traffic characterisation and presented an automatic method to extract and test android network traffic, but it did not quantify the ad traffic. [16] presented a novel redundancy elimination (RE) method to reduce the network traffic of smartphone by 30%. To the best of our knowledge, our paper is the first research to develop an automatic system to quantify the ad network traffic consumed by android apps.

III. Research Goal

The research goal of this work is twofold.

- Firstly, it aims at having a fine-grained measurement of ad network traffic. The existing work of packet capture software cannot precisely determine which packets belong to ad network. Modifying the android system to log the events of displaying ads can help to identify ad traffic, but it can not get detail information about ad traffic, such as packet size. In our work, we are able to acquire a fine-grained measure of ad network traffic by combining these two sources of traffic.
- Secondly, we aim at studying how different ad libraries affect the ad traffic. In particular, android apps usually contain more than one ad libraries and the developers may change the rate of displaying ads, which will lead to different bandwidth consumption speed. Without loss of generality, we developed our own android apps which have one ad library for one round of evaluation.

IV. System Design

To have a fine-grained classification of network traffic consumed by smartphone apps in android system, we develop an automatic analysis system which can automatically classify and quantify the ad traffic and malware traffic. In this section, we will introduce it in details.

A. Overview of Ad Analysis Platform

In order to extract the ad traffic, we develop an Ad Analysis Platform which is shown in Fig. 1. Once an android app is running on our modified android emulator, we use Router Level Capture and System Level Capture to capture the network traffic consumed by the app. We propose a novel traffic match algorithm to get the detail information about the ad traffic via analyzing these two sources of traffic packets. With these information, the system generates cluster rules to filter ad traffic. By filtering the captured traffic packets with these rules, we can get the quantity of ad traffic caused by the android app. The Ad Analysis Platform consists of the following four components: System Level Capture Component, Router Level Capture Component, Traffic Packet Clustering Component and Ad Traffic Quantification Component.

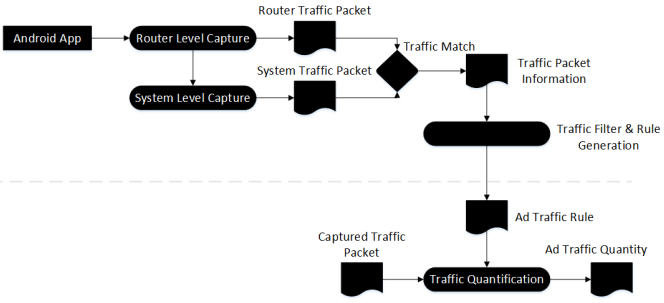

Fig. 1: Ad Analysis Platform.

- **System Level Capture Component**
  To achieve a fine-grained traffic capture, we need to characterize the ad traffic by capturing the traffic packet and its context through android system. Existing android system cannot accomplish this goal, so we decide to modify the android webview and log system. In this way, we are able to get a precise log of ad traffic.
- **Router Level Capture Component**
  We can capture the ad traffic precisely through System Level Capture Component. However, we are still facing the challenge of how to quantify the ad traffic since packet size remains unknown. To address this problem, we also capture all the traffic in the router level. Tshark can analyze the traffic packet and get packet size. By matching these two kinds of traffic, we are able to quantify the ad traffic.
- **Traffic Packet Clustering Component**
  We propose a novel traffic match algorithm for the sequence matching. With two kinds of traffic sources from System Level Capture and Router Level Capture, we use greedy match algorithm to extract ad traffic. The insight of the algorithm is to find the most similar traffic packets according to timestamp of packets in the two sources of traffic. Based on the proposed algorithm, we cluster the traffic and store them in Mongodb for the future use of traffic quantification.
- **Ad Traffic Quantification Component**
  We can label the ad traffic in the traffic packets captured

in the router level based on the packet clustering rules proposed in the above component. When the characteristics of a traffic packet match one of the clusters of ad library, we can label this traffic packet as ad traffic. After scanning all the TCP traffic packets, we can quantify the ad traffic precisely. If we run one app for a certain time, we can know the ad traffic consumed by the app.

B. Traffic Match Algorithm in Ad Analysis Platform

For ad traffic match problem, we abstract it into such a model. Given two sequences $\langle S \rangle$ = $(St_1,Sk_1),(St_2,Sk_2),...(St_n,Sk_n)$ and $\langle R \rangle$ = $(Rt_1,Rk_1),(Rt_2,Rk_2),...(Rt_m,Rk_m)$. $\langle S \rangle$ is a sequence of traffic packets from System Level Capture, and $\langle R \rangle$ is a sequence of traffic packets from Router Level Capture. (St_i,Sk_i) and (Rt_j,Rk_j) are traffic packets. St_i and Rt_j refer to time t_i, t_j of packet. Sk_i and Rk_j refer to related information of packet. The algorithm is a simple gready algorithm. For each $S_i \in \langle S \rangle$, we go through $\langle R \rangle$ to find a corresponding $R_j \in \langle R \rangle$ so that $|St_i - Rt_j| < 1$. Finally, we will get the most match sequence $\langle V \rangle$ = $(S_i,R_i)...(S_j,R_j)$ where $S_i, S_j \in \langle S \rangle$ and $R_i, R_j \in \langle R \rangle$ and $|St_i - Rt_i| < 1$.

Algorithm 1 Traffic Match Algorithm

```
1: Input: List A, List B
2: Output: A list of matching pair inside A and B
3: index = 0
4: for each a ∈ A do
5:     Find first index k that |a.time − B[k].time| < 1 in B[index...n]
6:     if Find Such k then
7:         index = k+1
8:     end if
9: end for
```

The insight of Algorithm 1 is to find the most similar traffic according to timestamp of the two sources of traffic. The running time complexity is $O(n^2)$. The algorithm is of high effectiveness proven by a series of experiments. After the ad traffic match, we are able to acquire some detailed information about the traffic packet, such as stack context and Uniform Resource Identifier (URI).

V. System Implementation

In this section, we introduce the implementation of Ad Analysis Platform. The Ad Analysis Platform is divided into four sections, router level capture, system level capture, traffic packet clustering and ad traffic quantification.

A. Router Level Capture Component

In this paper, we utilize Wireshark and Tcpdump to achieve Router Level Capture. Android Debug Bridge (adb) is a versatile tool that allows you to communicate with an emulator instance or connected android-powered device. We make use of adb to upload local file to android emulator, download the file in android emulator to local computer. We can also use it to install apps and check the device logs.

We just run a shell script. Then the app is installed in the android emulator and starts to run by itself without any human interactions. At the same time, the system captures all the network traffic through the android phone. After a period of time (e.g., 20 minutes), the capture process terminates and the app is uninstalled automatically. At the same time, the captured traffic packets have been stored in local computer.

B. System Level Capture Component

In System Level Capture, we need to characterize the ad traffic so that we can match the ad traffic in the Router Level Capture. To achieve the goal, we have to modify the android webview and log system.

After the ad library obtains the content of ad through HTTP request, it will invoke android webview widget to display the ad. Webview is a default widget provided by android system and it offers to display some webpage. When the ad library gets the uri which needs to be displayed, then the uri will be passed to the webview widget. Webview downloads all the resources related to the uri, including website, pictures and javascript codes. Then ads will be shown in the app with picture effects. To capture the specific traffic packets, we need to modify the webview widget. On the downloading of any new resource, the onLoadResource method of WebViewClient will be invoked. Therefore, we choose to rewrite the onLoadResource to label the ad traffic packets. The webview widget mechanism is shown as Fig. 2.

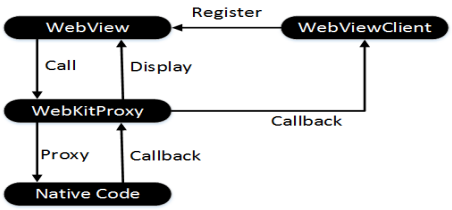

Fig. 2: Webview Widget.

After capturing the network traffic in system level, they need to be stored in database for future use of quantifying ad traffic. Since android system is multi-thread mechanism, storing the captured traffic packets may encounter conflicts. Android system provides a log system which has a mutual log mechanism guaranteeing that log files can be written without interruption. However, the log is limited to 1024 bytes and we need much more than 1024 bytes to record the contexts of the traffic packets. Besides, the log system is designed for android applications and cannot be invoked in android kernel. Therefore, we have to modify the log system.

The System Level Capture is illustrated as Fig. 3. The traffic packet and its context are captured together. After serializing, the json file is generated which is more than 2000 bytes and it is beyond the limitation of the log system. To solve this problem, we split the file into several files with the same tag. Different json files have different tags. Having recorded all the

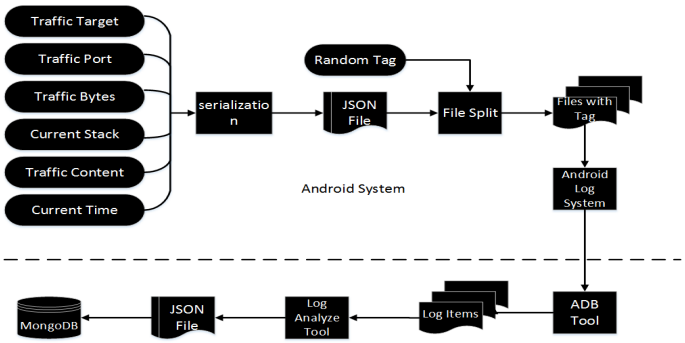

Fig. 3: System Level Capture Process.

logs, we need to extract the log information from the android emulator with the help of ADB tool. We design an automatic log extraction system. The system monitors the output of the ADB log and accumulates the same log information. After all these processes, the traffic packets from System Level Capture are stored in MongoDB.

C. Traffic Packet Clustering Component

We will get two sources of traffic packets from Router Level Capture and System Level Capture. With the help of our traffic match algorithm, we can obtain some detailed information about traffic packets, such as stack context and URI. By comparing the keyword of ad library with URI, we can verify whether a traffic packet belongs to ad traffic. Having got the information about ad traffic, we need to cluster the ad traffic and quantify them. We divide the URI into three distinguished

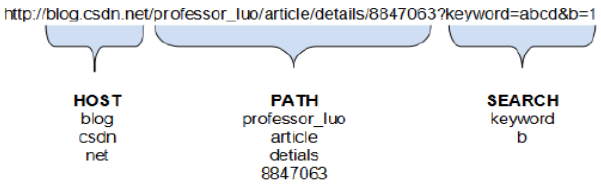

Fig. 4: Structure of URI

parts, host address, URI path and query string, as is shown in Fig. 4. Thus, we can acquire three different characterisations of a URI. By utilizing these characterisations, we are able to calculate the similarity of any two URIs. We cluster traffic packets with the similarity of URIs.

$$\begin{aligned} Dis(U_a, U_b) = (&DisSet(U_a.Path, U_b.Path) \\ &+ DisSet(U_a.Query, U_b.Query) \\ &+ DisSuf(U_a.Host, U_b.Host))/3 \end{aligned} \tag{1}$$

$$DisSet(A, B) = 1 - \frac{A \cap B}{A \cup B} \tag{2}$$

$$DisSuf(A, B) = 1 - \frac{MS(A, B)}{L(A) + L(B) - MS(A, B)} \tag{3}$$

$Dis(U_a, U_b)$ means the distance between URI a and URI b and query is the search part in the URI. The method $DisSet$ represents the distance of two sets and $DisSuf(A, B)$ is the distance of two hosts. $MS(A, B)$ means max suffix between host A and host B. $L(A), L(B)$ are lengths of host A and host B. We make use of longest common suffix to measure the host characterisation of two URIs. For instance, there is a service with two servers and the websites are aws1.sina.com.cn and aws2.sina.com.cn. There is a common string of length 3 in both of the two URIs, then the similarity is 0.6 and distance is 0.4. Therefore, if the service is provided by the same company, the similarity is relatively high. It is reasonable to measure the similarity of host with longest common suffix.

Having calculated the similarity of any two URIs, we begin to cluster these URIs. At fist, every URI is regarded as a single cluster, and distance of two clusters is defined as the longest distance between any two URIs in the cluster. If the distance of any two clusters is smaller than a specific distance, we combine these two clusters to form a new cluster. Finally, we can obtain several clusters of ad traffic.

D. Ad Traffic Quantification Component

With the clusters of ad traffic, we have the ability to quantify the ad traffic from captured traffic packets in Router Level Capture. For any HTTP packet, we verify whether the packet belongs to one of the ad traffic clusters, namely, whether the largest distance between the URI in the packet and the URI in the ad traffic cluster is less than a specific threshold. It is defined as Equation (4).

$$\begin{aligned} &MinMaxDis(URI, u), u \in Cluster_j, \\ &Cluster_j \in Clusters \leq M \end{aligned} \tag{4}$$

We can get the best result with M assigned as 0.6 in a series of experiments. When certain traffic packet belongs to the ad traffic clusters, the traffic packet is labeled as ad traffic. However, only HTTP packet is not enough to cover the whole ad traffic. HTTP is based on TCP and TCP needs three-hand shakes to establish a connection which also consumes much traffic. Besides, when transfering pictures, the file is transfered by splited packets and each packet needs to cause corresponding traffic. These traffic will not be accounted if we only consider HTTP packet. In this case, we have to take the TCP packets into account. It's easy to acquire the host address of the TCP packets with its corresponding HTTP packet. In this way, we are able to quantify all the ad traffic of a certain app.

VI. Evaluation

A. Evaluation of Matching Algorithms

We randomly selected 10 ad providers. According to Appbrain [10], admob and millennial take a majority of app market and reach a percentage about 50% of the market. It is difficult to validate traffic match algorithm with randomly selected android apps since they not only cause ad traffic URIs, but also account for some other URIs, like URIs for services provided by the app. In this situation, we use the ad library Software Development Kit (SDK) to develop our own android apps so that they can only generate ad traffic URIs. Under

Ad Traffic(KB)	Quantity	Rate
$\geq$ 100	51	92.7%
$\geq$ 200	43	78.2%
$\geq$ 400	32	58.2%
$\geq$ 600	16	29.1%
$\geq$ 800	7	12.7%
$\geq$ 1000	3	5.5%

(a) Noraml Apps

Ad Traffic(KB)	Quantity	Rate
$\geq$ 100	18	100%
$\geq$ 200	15	83%
$\geq$ 400	10	55%
$\geq$ 600	7	38%
$\geq$ 800	7	38%
$\geq$ 1000	4	22%
$\geq$ 1200	2	11%

(b) Repacked Apps

Ad Traffic(KB)	Quantity	Rate
$\geq$ 20	19	100%
$\geq$ 40	16	84%
$\geq$ 60	13	68%
$\geq$ 80	3	15%
$\geq$ 100	1	5%

(c) Malicious Apps

Fig. 5: Ad Traffic of Android Apps

this circumstance, we use them to validate our traffic match algorithm.

TABLE I: Traffic Match Result

Ad Library	URI	Match
admob	51	51
millennial	287	287
youmi	117	117
mobclix	115	113
jumptap	92	90
mopub	88	88
mobfox	48	47
adwo	43	42
leadbolt	32	32
cauly	27	27

From Table I, we can see a high match rate. We have matched 894 URIs in 900 URIs and the match rate is more than 99.33%. Then we take admob for example to show our clustering result. In 51 HTTP traffic packets which contain URIs, we find there are 7 clusters. The front 4 clusters have 20 URIs in total and the distances in these clusters are zero for they are all the same URIs. Another 3 clusters contain 31 URIs. The 31 URIs are all like http://com.googleads/, http://googleads.g.doubleclick/ and they are all websites of ads from Admob server. The evaluation results show that the proposed clustering algorithm works well.

B. Ad Traffic Analysis

In the following parts, we introduce the ad traffic of normal apps, repacked apps and malicious apps. Then we show the ad library mechanism adopted by current pupular ad libraries.

1) Ad Traffic of Normal Apps: In order to quantify ad traffic of normal apps, we downloaded 1000 free apps from Google Play. Without loss of generality, we selected 40 apps from 25 categories respectively. In the 1000 apps, we find that more than half free apps are embedded with ad libraries and admob takes the majority. More than 14% apps have more than one ad libraries.

We randomly selected 55 apps with ad libraries for the experiment. We ran each app for 20 minutes and quantified the ad traffic consumed by each of them. From Fig. 5(a) we can see more than half apps cause more than 400 KB network traffic in merely 20 minutes. Besides, the ad traffic takes more than 80% of total traffic for most apps. It means that the majority of network traffic users pay for is useless and not necessary,

2) Ad Traffic of Repacked Apps: To quantify the ad traffic of repacked apps, we acquired 320 repacked apps from PANGU research lab. We randomly selected 19 apps with ads from those apps. Similar to normal apps we can see from Fig. 5(b), more than half repacked apps consumed network traffic over 400 KB in merely 20 minutes. Besides, the ad traffic causes more than 80% of total traffic for most apps.

Meanwhile, we downloaded 9 original apps corresponding to the repacked apps. The Fig. 6 shows that repacked apps cause more network traffic than original apps relatively. It explains why the malicious developers prefer to repack the apps by embeding other ad libraries into the original ones, because the more frequently ads are seen by users, the more money they will earn in return.

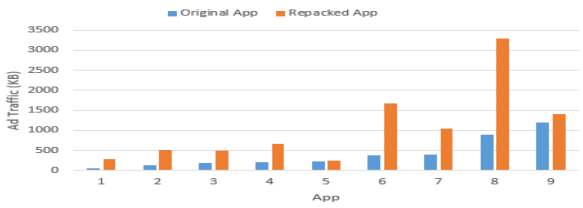

Fig. 6: Ad Traffic of Original and Repacked Apps

3) Ad Traffic of Malicious Apps: To quantify the ad traffic of malicious apps, we obtained a malicious app dataset from Android Malware Genome Project[12]. Different from normal apps and repacked apps, malicious apps consumed rather small amount of network traffic. Fig. 5(c) shows that most apps caused less than 100 KB in 20 minutes. However, for most malicious apps, the only network traffic caused by the app is ad traffic in 20 minutes.

C. Ad Library Mechanism Analysis

There are two kinds of mechanisms for ad service providers, which determines how to load ads from remote servers.

- The mechanism 1 is to download all the ads from the remote server at the time and store them in the local database. The app just needs to fetch the ad from local database to display without consuming extra network traffic.

- The mechanism 2 is to fetch the ad from the remote server each time the ad is displayed. By this means, extra network traffic will be caused.

From the statistics of google play we collect, many android apps contain more than one ad libraries. We select 10 popular ad libraries of android market. To investigate how it works, we develop our own android app with these ad libraries and each app demo with only one ad library and the same setting. We ran the apps for 60 minutes and the data was cut into six parts with each part 10 minutes. During the experiment, the apps either displayed the same ads or several repeated ads.

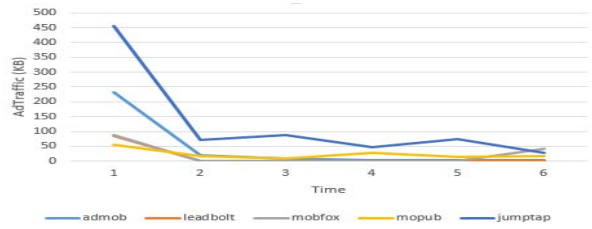

Fig. 7: Ad Library Mechanism 1

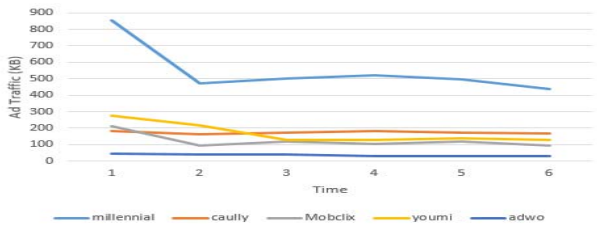

Fig. 8: Ad Library Mechanism 2

Comparing ad traffic consumed by each ad library, we can divide these ad libraries into two classes. From Fig. 7 and Fig. 8, it is observed that admob, leadbolt, mobfox, mopub, jumptap take Mechanism 1 and millennial, caully, mobclix, youmi and adwo take Mechanism 2. Mechanism 1 causes a high traffic overhead in first 10 minutes and cause rather small amount of traffic in the following 50 minutes. While mechanism 2 consumes almost the same level amount of ad traffic all the time and these is no big difference between the data in first 10 minutes and the following 50 minutes.

VII. Conclusion

In this paper, we develop an automatic ad analysis platform by modifying the android system to quantify the ad traffic of android app. The ad analysis platform characterises the ad traffic and then successfully quantifies the ad traffic. We test three different kinds of apps. Most normal apps and repacked apps consume more than 400 KB ad network traffic in 20 minutes and some even exceed 1M. It is surprising that the malicious apps cause small amount of ad traffic relatively. We also find that the ad library mechanisms of some popular ad libraries are different. During the experiment, half ad libraries prefetch the ads and store them in local database. The other half have to fetch the ads each time the ads are shown to the users.

VIII. Acknowledgement

This work is supported by National High-Tech R&D (863) Program (no. 2015AA01A707)

References

[1] CISCO Report, http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html
[2] Guha, Saikat, Bin Cheng, and Paul Francis. "Privad: practical privacy in online advertising." In USENIX conference on Networked systems design and implementation, pp. 169-182. 2011.
[3] Grace, Michael C., Wu Zhou, Xuxian Jiang, and Ahmad-Reza Sadeghi. "Unsafe exposure analysis of mobile in-app advertisements." In Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks, pp. 101-112. ACM, 2012.
[4] Book, Theodore, Adam Pridgen, and Dan S. Wallach. "Longitudinal analysis of android ad library permissions." arXiv preprint arXiv:1303.0857 (2013).
[5] The Economist, http://www.economist.com/news/business/21635497-dark-corner-digital-advertising-business-needs-cleaning-up-dial-b-bot
[6] Falaki, Hossein, Dimitrios Lymberopoulos, Ratul Mahajan, Srikanth Kandula, and Deborah Estrin. "A first look at traffic on smartphones." In Proceedings of the 10th ACM SIGCOMM conference on Internet measurement, pp. 281-287. ACM, 2010.
[7] Shekhar, Shashi, Michael Dietz, and Dan S. Wallach. "Adsplit: Separating smartphone advertising from applications." In Presented as part of the 21st USENIX Security Symposium (USENIX Security 12), pp. 553-567. 2012.
[8] Tongaonkar, Alok, Shuaifu Dai, Antonio Nucci, and Dawn Song. "Understanding mobile app usage patterns using in-app advertisements." In International Conference on Passive and Active Network Measurement, pp. 63-72. Springer Berlin Heidelberg, 2013.
[9] Dai, Shuaifu, Alok Tongaonkar, Xiaoyin Wang, Antonio Nucci, and Dawn Song. "Networkprofiler: Towards automatic fingerprinting of android apps." In INFOCOM, 2013 Proceedings IEEE, pp. 809-817. IEEE, 2013.
[10] Appbrain ad library, http://www.appbrain.com/stats/libraries/ad
[11] PANGU research lab, http://en.pangu.io/
[12] Android Malware Genome Project, http://www.malgenomeproject.org/
[13] Lee, Kyunghan, Joohyun Lee, Yung Yi, Injong Rhee, and Song Chong. "Mobile data offloading: how much can WiFi deliver?." In Proceedings of the 6th International COnference, p. 26. ACM, 2010.
[14] Perdisci, Roberto, Wenke Lee, and Nick Feamster. "Behavioral Clustering of HTTP-Based Malware and Signature Generation Using Malicious Network Traces." In NSDI, pp. 391-404. 2010.
[15] Xu, Kuai, Feng Wang, and Lin Gu. "Behavior analysis of internet traffic via bipartite graphs and one-mode projections." IEEE/ACM Transactions on Networking 22, no. 3 (2014): 931-942.
[16] Qian, Feng, Junxian Huang, Jeffrey Erman, Z. Morley Mao, Subhabrata Sen, and Oliver Spatscheck. "How to reduce smartphone traffic volume by 30%?." In International Conference on Passive and Active Network Measurement, pp. 42-52. Springer Berlin Heidelberg, 2013.
[17] Zhang, Li, Dhruv Gupta, and Prasant Mohapatra. "How expensive are free smartphone apps?." ACM SIGMOBILE Mobile Computing and Communications Review 16, no. 3 (2012): 21-32.
[18] Li, Huaxin, Zheyu Xu, Haojin Zhu, Di Ma, Shuai Li, and Kai Xing. "Demographics inference through Wi-Fi network traffic analysis." In IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, pp. 1-9. IEEE, 2016.
[19] Li, Muyuan, Haojin Zhu, Zhaoyu Gao, Si Chen, Le Yu, Shangqian Hu, and Kui Ren. "All your location are belong to us: Breaking mobile social networks for automated user location tracking." In Proceedings of the 15th ACM international symposium on Mobile ad hoc networking and computing, pp. 43-52. ACM, 2014.
[20] Zhu, Haojin, Suguo Du, Zhaoyu Gao, Mianxiong Dong, and Zhenfu Cao. "A probabilistic misbehavior detection scheme toward efficient trust establishment in delay-tolerant networks." IEEE Transactions on Parallel and Distributed Systems 25, no. 1 (2014): 22-32.