Understanding and Detecting Mobile Ad Fraud Through the Lens of Invalid Traffic

Suibin Sun
Shanghai Jiao Tong University
sun1998@sjtu.edu.cn

Le Yu
Shanghai Jiao Tong University
yule5100309221@sjtu.edu.cn

Xiaokuan Zhang
The Ohio State University
zhang.5840@osu.edu

Minhui Xue
The University of Adelaide
jason.xue@adelaide.edu.au

Ren Zhou
Shanghai Jiao Tong University
zhouren@sjtu.edu.cn

Haojin Zhu*
Shanghai Jiao Tong University
zhu-hj@sjtu.edu.cn

Shuang Hao
University of Texas at Dallas
shao@utdallas.edu

Xiaodong Lin
University of Guelph
xlin08@uoguelph.ca

ABSTRACT

Along with gaining popularity of Real-Time Bidding (RTB) based programmatic advertising, the click farm based invalid traffic, which leverages massive real smartphones to carry out large-scale ad fraud campaigns, is becoming one of the major threats against online advertisement. In this study, we take an initial step towards the detection and large-scale measurement of the click farm based invalid traffic. Our study begins with a measurement on the device's features using a real-world labeled dataset, which reveals a series of features distinguishing the fraudulent devices from the benign ones. Based on these features, we develop EvilHunter, a system for detecting fraudulent devices through ad bid request logs with a focus on clustering fraudulent devices. EvilHunter functions by 1) building a classifier to distinguish fraudulent and benign devices; 2) clustering devices based on app usage patterns; and 3) relabeling devices in clusters through majority voting. EvilHunter demonstrates 97% precision and 95% recall on a real-world labeled dataset. By investigating a super click farm, we reveal several cheating strategies that are commonly adopted by fraudulent clusters. We further reduce the overhead of EvilHunter and discuss how to deploy the optimized EvilHunter in a real-world system. We are in partnership with a leading ad verification company to integrate EvilHunter into their industrial platform.

CCS CONCEPTS

• **Security and privacy** → *Software and application security*; • **Networks** → Network measurement.

*Haojin Zhu (zhu-hj@sjtu.edu.cn) is the corresponding author.

CCS '21, November 15–19, 2021, Virtual Event, Republic of Korea

ACM ISBN 978-1-4503-8454-4/21/11…$15.00
https://doi.org/10.1145/3460120.3484547

KEYWORDS

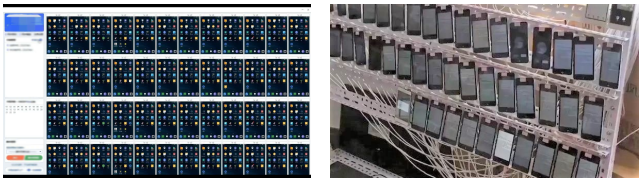
Invalid Traffic; Ad Fraud; Click Farm

ACM Reference Format:
Suibin Sun, Le Yu, Xiaokuan Zhang, Minhui Xue, Ren Zhou, Haojin Zhu, Shuang Hao, and Xiaodong Lin. 2021. Understanding and Detecting Mobile Ad Fraud Through the Lens of Invalid Traffic. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15–19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3460120.3484547

1 INTRODUCTION

Programmatic advertising (ad) has become the driving force for the growth of internet advertising in the past decades, which is benefited from the innovation of new ad technologies that bring us the efficiency of automatic buying and selling of advertisements. Real-Time Bidding (RTB), one of the most significant technologies, provides a digital marketplace, where website owners and mobile app developers (ad publishers) can sell the spare spaces (ad slots) on their websites or applications to advertisers through real-time auctions. Over the past 10 years, mobile in-app advertising has become the major increasing growth of advertising revenues. It is reported that the global RTB market size could potentially grow from $6.6 billion USD in 2019 to $27.2 billion USD by 2024 [33].

Unfortunately, this increased prominence has also attracted the attention of fraudulent ad slot sellers, who try to inflate their incomes by fabricating ad requests. This task is usually outsourced to click farm owners, who can control thousands of mobile devices or emulators and mimic normal app usage patterns to trick advertisers into believing that their ads have been seen by actual, interested users. Fig. 1 substantiates an example of a "click farm" reported by the recent news [26, 32], which leverages massive real smartphones to carry out a large-scale ad fraud campaign. Nowadays, an advertiser usually pays money for every 1,000 impressions of a given advertisement, instead of clicks, which is commonly known as Cost Per Mille (CPM) [20]. As a result, those committing frauds within this system have evolved their technique from "click spam" into the generation of "invalid traffic". It is estimated that every year trillions of delivered ad impressions are not watched by real people, leading to losses in the tens of billions of US dollars for advertisers [12, 42].



(a) Software level emulation (b) Hardware level automation

Figure 1: Automated and coordinated "click farm".

Due to the substantial financial loss caused by invalid ad traffic, the problem has attracted increasing attention. However, many prior works that focus on the problem of click spam detection [13, 14, 40] or authenticated click [15, 21, 36] cannot be applied in the era of RTB, since the basic pricing model in the RTB system is CPM rather than traditional CPC (cost-per-click), indicating that fraudsters no longer need to click on the ads, which renders detecting the invalid traffic more difficult.

In the industry, traffic verification is performed by designated companies/organizations, which are trusted by both ad publishers and advertisers. By removing the invalid traffic before the billing cycles, they help advertisers minimize wrongful payment while protecting the rights of ad publishers. The key players in this ecosystem include Integral Ad Science (IAS) [3], Oracle [4], and White Ops [5]. The best practice of combating invalid traffic adopted by the mainstream traffic verification companies heavily relies on rule-based detection such as blacklists and parameter/metadata checks (*e.g.*, IP blacklists, location validity checking, user agent (UA) checking). To date, little attention has been paid to the detection and measurement of large-scale click farms.

Challenges. Detection of click farms faces the following unique challenges. 1) Attackers have adopted a series of hardware/software level strategies (*e.g.*, frequently changing the various parameters including IP, UA, IMEI, emulating the human behavior including the mobility pattern, and even using the automated tool) to mimic the human behaviors and generate "seemingly organic" traffic, which is difficult to detect. 2) Due to the noisy ad traffic data and dynamic network environment, the detection of invalid traffic based on any specific fraudulent device is less reliable. 3) In practice, it is expected to process billions of transactions per day. The amount of data that needs to be processed makes it challenging to design a practical detection system with low overhead.

EvilHunter. In this study, we present EvilHunter, the first work to investigate click farm-oriented invalid traffic detection based on the real-world mobile RTB transaction data. EvilHunter handles invalid traffic via detecting the source of traffic, *i.e.*, the fraudulent devices involved in the click farms. EvilHunter is motivated by the following observation: although any individual fraudulent device tends to adopt strategies to mimic the organic traffic, the attackers behind the click farm have strong incentives to camouflage the behaviors of invalid traffic with certain patterns in order to lower the cost. Such group features could be used to design a novel automatic click-farm and fraudulent device detection scheme.

Based on the above insights, we first conduct a measurement of the device's features using a real-world labeled dataset, which reveals a series of features distinguishing the fraudulent devices from the benign ones. To make use of the features depicted by fraudulent devices as groups, we further propose a three-stage detection system, EvilHunter: 1) Stage 1 uses the identified features to build a classifier to flag individual devices; 2) Stage 2 captures cluster-level features by applying the *Top-App based Clustering Algorithm*, which leverages the app usage patterns of the devices to group devices; 3) Stage 3 aggregates the information produced in the previous stages and performs majority voting to detect click farms as well as the fraudulent devices.

Our extensive evaluation shows that EvilHunter is able to capture real-world click farms with high accuracy and with moderate overhead incurred. We also reveal interesting findings surrounding the detected click farms, which may benefit future research. We have reported all of the findings to a leading ad verification company (Company A), which has positively acknowledged our results. Based on our study, we have contributed a fraud reason code on click farm detection in Company A's real-world detection system, which is expected to help the ad verification industry to identify large-scale fraudulent device clusters and filter invalid traffic.

Contributions. We make the following key contributions:

- *New features (Sec. 4).* We perform an ad fraud measurement study on a labeled real-world dataset, which reveals a series of important characteristics of fraudulent devices.
- *New system (Sec. 5).* We propose EvilHunter, a novel three-stage fraud detection system for automatically identifying device clusters and classifying the fraudulent devices in terms of their cluster-level features based on real-world ad bid logs.
- *New findings on cheating strategies of click farms (Sec. 7).* We successfully identify a group of large-scale click farms in the real-world dataset. After focusing on the largest click farm and tracing back to the historical data in two 10-day datasets in 2018 and 2019, respectively, we discover similar patterns in a large number of devices. We reveal a series of strategies adopted by this super click farm to evade detection.
- *Optimization and real-world deployment (Sec. 8).* We propose several optimization mechanisms that greatly reduce the overhead of EvilHunter, and make it practical to be used in the real world. We evaluate EvilHunter on a 1-day unlabeled dataset containing 53M devices. Our evaluation shows that EvilHunter is able to detect 8M fraudulent devices related to click farms within 2 hours. The top results are confirmed by Company A, and Company A has integrated EvilHunter into its real-world system.

Ethical considerations. Each time a user's device requests an ad material, the ad exchange will transmit the request and record it as a log. The source data in the request is collected by the ad software development toolkit (SDK) embedded in an app only after the user's consent on the app's privacy agreement. The ad traffic verification company routinely collects ad bid request logs from ad exchange for the purpose of verifying and measuring the quality of the ad traffic. The data is kept in the ad traffic verification company data center with access being granted only to the authors' affiliation. We have obtained approval from the ad traffic verification company for accessing the ad bid logs. The data (such as IMEIs) provided by

the ad traffic verification company does not include any Personal Identifiable Information (PII).

2 BACKGROUND

In this section, we introduce the system model of programmatic advertising and then explain the ecosystem of real-time bidding by showing a typical process of serving an ad in programmatic advertising and the role of fraudulent publishers. We also discuss the best practice of invalid traffic (IVT) verification and filtering in the industry.

Ecosystem of programmatic advertisement. Digital advertising is mainly processed over "programmatic" platforms in an automatic way, which involves *Advertiser*, *Publisher*, *Demand Side Platform (DSP)*, *Supply Side Platform (SSP)*, and *Ad-Exchange*. The Publisher (*e.g.*, developers of mobile apps) reserves space in their apps as ad slots for displaying advertisements and uses SSPs to auction their available ad slots. Advertisers use DSPs (*e.g.*, TheTradeDesk and Baidu) to bid on these available ad slots based on how successful they think those ads will be in attracting the interest of visitors. Similar to the stock exchange, the *Ad-Exchange* (*e.g.*, Google, MoPub, and Tencent) serves as a digital marketplace, which allows advertisers and publishers to sell and buy ad slots through RTB-based auctions. Each ad slot would typically go through many auctions before being matched to the final advertiser. These auctions happen billions of times per day, usually within the milliseconds before an ad is presented on the end user's device.

A typical RTB process is illustrated in Fig. 2. When a user opens up an app and triggers an ad impression (*i.e.*, a click on the ad), the app sends an ad request to the SSP, which further forwards the request to the Ad-Exchange. The ad request often contains device ID, IP address, timestamp, the user's current location, as well as other information about the ad slot. Upon receiving the ad request, the Ad-Exchange initiates an auction of the ad slot and records an ad bid log in the database. The bidding request is then sent to each registered DSP in the Ad-Exchange and the DSPs bid on the ad slot on behalf of the advertisers. As specified by OpenRTB [20], in the most real-time bidding systems, Cost Per Mille (CPM) is used to charge advertisers. When the Ad-Exchange initiates an auction, the DSPs on behalf of advertisers would respond to the optimal price, in accordance with the rules set by advertisers. The Ad-Exchange informs the winner, and the corresponding advertisement is then sent to the user and is displayed on the app.

Ad fraud. Unfortunately, this increased prominence has also attracted the attention of tech-savvy fraudulent publishers, who try to produce fake traffic and fraudulent ad inventories to trick advertisers into believing that their ads are being seen by genuinely interested users. According to the Invalid Traffic (IVT) Detection and Filtration Standards [24], IVT can be caused by multiple sources. On the one hand, IVT involves the traffic identified through routine means of filtration, executed through the application of lists, or with other standardized parameter checks. This type of IVT can be generated by known invalid data-center traffic, bots, spiders, or other crawlers, non-browser agents. On the other hand, some types of IVT, produced by known automated systems, emulators, custom automation software and tools, infected and hijacked devices, incentivized human activities, and adware/malware that conducts

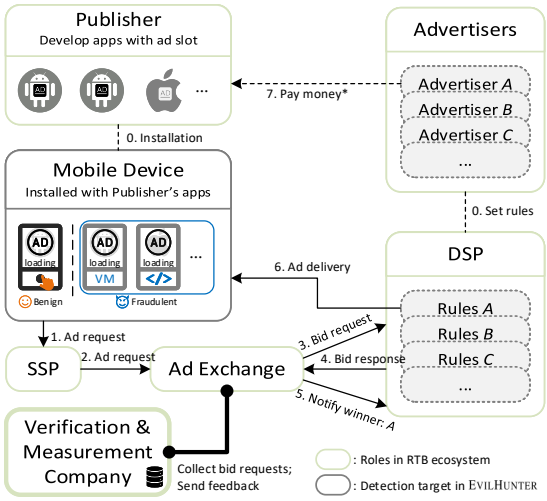


Figure 2: The ecosystem of "programmatic" advertising and the flows of a typical RTB process. The payment process indexed in 7 is achieved indirectly via the intermediates.

deceptive actions (*e.g.*, ad injection and unauthorized overlays), require more advanced techniques to analyze and detect, which often include multi-point corroboration/coordination and a significant amount of human intervention. In this work, we focus on detecting the provenance of IVT (*i.e.*, the fraudulent devices), and discover organized device clusters.

Ad traffic validation. To defend against ad fraud, the industry relies on third-party ad traffic verification services provided by the ad traffic verification companies, such as White-Ops, IAS, and RTBAsia. The traffic identified as invalid must be deducted from profits between the publisher and the advertiser as per the Invalid Traffic Detection and Filtration Standards [24]. To enhance the robustness of the detection results, there is a pressing need to compare and cross-check the IVT measurement results across different measurement organizations. For example, 30 major IVT verification companies across Asia-Pacific, including RTBAsia, Baidu, Tencent, and Toutiao, carry out the project of Distributed Invalid Traffic Filter (DIF). This allows different traffic validators to upload their own IVT measurements based on billions of daily ad bid request records and vote if there is any specific suspicious device or IP belonging to the attackers. Therefore, it is desirable to design a novel third-party traffic validation approach by monitoring the ad bid request logs.

3 DATASETS

In this section, we present the format of the ad bid logs, which serve as the basis of our design of EvilHunter. We also introduce the information of the datasets used in the paper.

3.1 The Format of Ad Bid Logs

Our datasets contain the ad bid logs collected by our industrial collaborators, which record the ad bid requests in ad exchange platforms. A typical ad bid request contains 10 fields (see Table 1). The `IP` refers to the IP address of the device. The `Ad Slot ID` represents the unique identifier of the ad slot assigned by the ad-exchange platform. There are three types of Device IDs: `IMEI`, `Android ID`,

Table 1: Fields of ad bid logs.

Field	Description
IP	Source IP
Ad Slot ID	A globally unique id of the requested ad slot
IMEI	IMEI MD5 value
Android ID	Android ID raw/MD5 value
IDFA	IDFA MD5 value
OS	Operating system of the device
Location	Real-time GPS coordinates of the device
Timestamp	The time when the request was sent
Bundle ID	Bundle ID of the app generating the request
Device Brand	The brand of the device
User-Agent (UA)	The user agent of the http request

Table 2: Datasets used in this paper; D_{train} and D_{test} are extracted from D_{2020}.

Name	Labeled?	Devices		Log	Duration	Used in	Purpose
		F	B				
D_{2020}	Y	2M	0.2M	82M	May 6 - June 5, 2020	Sec. 4	Fraudulent devices measurement
D_{train}	Y	120k	113k	4.3M	May 6 - May 15, 2020	Sec. 6	Training of EvilHunter
D_{test}	Y	125k	124k	4.9M	May 16 - May 25, 2020	Sec. 6 Sec. 7	EvilHunter evaluation Click farm investigation
D_{2018}	N	290M		290M	Mar 21 - Mar 30, 2018	Sec. 7	Backtracking the largest click farm; Cheating strategy investigation
D_{2019}	N	63M		63M	Mar 6 - Mar 15, 2019		
D_{2021}	N	53M		117M	Jan 13, 2021	Sec. 8	In-the-wild detection & validation

and IDFA. All of them are hashed by MD5. IMEI and Android ID are used in Android devices while IDFA is for iOS devices. The OS indicates the operating system of the mobile device, either Android or iOS. The Location represents the device's geo-location at the time of ad request generation. The Timestamp refers to the time when the request is sent. The Bundle ID indicates from what app the ad request is originated. The Device Brand represents the brand of the device. The User-Agent (UA) refers to the user agent of the ad HTTP request. All the fields, except for IP and Timestamp, are reported in ad request parameters by the app.

3.2 Overview of Datasets

We use 6 different datasets in this paper, which are described in Table 2. All the datasets contain ad bid logs generated by mobile devices during a certain period. Here we briefly introduce each dataset, and more details will be provided in later sections where they are in use.

D_{2020}. D_{2020} is a labeled dataset containing ad bid request logs recorded from May 6, 2020, to June 5, 2020. The dataset contains 82 million logs, generated by 2 million *fraudulent* devices and 0.19 million *benign* devices. We use it as a ground-truth dataset to find distinct features for fraudulent devices (Sec. 4).

D_{train} and D_{test}. D_{train} and D_{test} are extracted from D_{2020}. They serve as a training set and a test set for the evaluation of EvilHunter (Sec. 6). Moreover, D_{test} is used to perform click farm investigation in Sec. 7.

D_{2018} and D_{2019}. After identifying a few click farms in D_{test}, we pick the largest click farm and trace back to two 10-day datasets in 2018 and 2019 to identify devices that share the same characteristics (Sec. 7). These click farm-related devices in 2018 and 2019 form D_{2018} and D_{2019}. We use these two datasets to investigate the general cheating strategies of click farms (Sec. 7).

D_{2021}. To evaluate the practicality to deploy EvilHunter in the real world, we use D_{2021} as a validation dataset, which contains 1-day's data without labels in 2021.

Ground-truth labels. In D_{2020}, the fraudulent devices are collected from a distributed blockchain system, where a group of leading industrial traffic verification companies work together to report highly suspicious devices. They identify the suspicious devices with auxiliary information collected by their own SDKs. Each of these companies regularly uploads fraudulent device information found by itself to this blockchain platform for majority voting: one device is deemed to be fraudulent if more than two members upvote it, and it will be added to the blacklist. Each blacklisted device will be blocked by them for several months, until being removed from the blacklist after a certain time. On the other hand, benign devices are collected by Company A using some incentives to encourage users to upload some evidence (*e.g.*, photos of surroundings) to prove that they are human. These pieces of evidence are examined by Company A manually to ensure that they are real. Both fraudulent and benign devices are double-checked by Company A's commercial rule-based system, which takes into consideration other aspects of the devices, such as the account activeness of the device in social media, the physical trace of the device. We extract the involved ad bid logs of the fraudulent devices and benign devices as dataset D_{2020}.

4 MEASURING FRAUDULENT DEVICES

Ad bid requests not only record the ad transaction history between the organic (benign) users but also serve as snapshots of evidence related to ad fraud. This provides us with the opportunity to capture the fraudulent devices and screen out the invalid traffic. In this section, we first use a real-world ad bid log dataset (D_{2020}) to study the features of fraudulent devices. These measurement results serve as the basis of EvilHunter. It is observed that fraudulent devices exhibit different patterns, *e.g.*, they are likely to adopt more IPs to generate ad bid requests for one or two ad slots. Here we take several examples to show the differences between fraudulent devices and benign devices as shown in Fig. 3.

Statistical Number: # unique IP addresses.

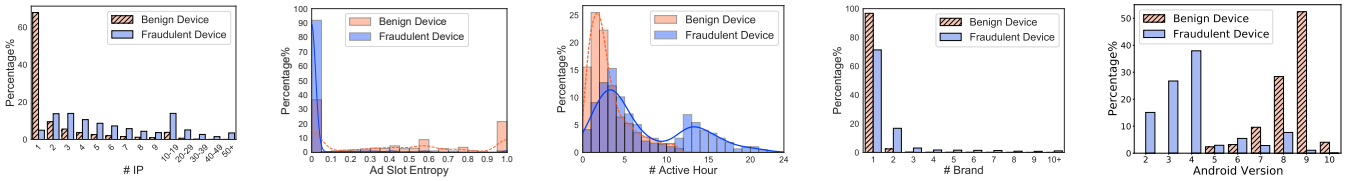
Observation 1: Fraudulent devices bind to more IP addresses.

The numbers of unique IP addresses used by fraudulent devices and benign devices are shown in Fig. 3a. As seen from Fig. 3a, fraudulent devices bind to more IPs than benign ones. 67.9% of benign devices use 1 IP address, and 4.9% of them use more than 10 different IP addresses. In contrast, 48.0% of fraudulent devices use more than 5 different IP addresses and 3.5% of them correspond to 50 different IPs. This phenomenon is perhaps due to the extensive usage of commercial residential IP proxy services, well aligned with the previous study [25].

Entropy: ad slot IDs.

Observation 2: Fraudulent devices have lower ad slot entropy.

We compare the entropy of ad slot IDs in Fig. 3b. As shown in Fig. 3b, 91.9% of fraudulent devices have an entropy of 0, meaning that they only had one unique ad slot ID. Intuitively, the fraudulent devices target the specific ad slot ID to make profits. However, benign devices usually request more than one ad slot IDs to enjoy the various



(a) # of unique IPs. **(b) Entropy of ad slot IDs.** **(c) Active hours per day.** **(d) # of device brands.** **(e) Android OS version.**

Figure 3: The comparison of different features between fraudulent devices and benign devices in D_{2020}. The lines in Fig. 3b and Fig. 3c represent kernel density estimations across devices.

services. This leads to an interesting phenomenon that benign users have a higher value in terms of ad slot entropy. It is observed that there are 41.9% benign devices with an entropy larger than 0.5, in contrast to the fraudulent devices with the proportions of 2.2%.

Temporal: active hours.

Observation 3: Fraudulent devices are more active.

We extract the active hours for the devices in both labels. From Fig. 3c, we can learn that most of the benign devices (99.9%) have less than 12 active hours per day in the dataset. In contrast, there are 11.0% of the fraudulent devices being active for more than 15 hours per day in the dataset. Even worse, 1.5% of them are active for more than 20 hours per day, which is unbelievable for humans. The potential reason behind this is that the attackers exhibit a high incentive to generate more invalid traffic within a specific period to gain more economic revenue.

Inconsistency: # device brand names.

Observation 4: Fraudulent devices use multiple brand names while benign devices usually use one brand.

The number of device brands is shown in Fig. 3d. More than 95% of benign devices only used one brand name regardless of the datasets; only 16.8% used two brand names. However, 16.8% of fraudulent devices used two brand names; roughly 5.6% of them used more than 5 brand names. More brands for fraudulent devices may occur when attackers frequently change the device's brand in lieu of device IDs.

Android version.

Observation 5: Fraudulent Android devices run lower OS versions.

As shown in Fig. 3e, we observe that fraudulent devices are installed on lower Android versions in comparison to benign devices. 9% (resp. 84%) of fraudulent (resp. benign) devices are on Android 8, 9, and 10. 79.7% of fraudulent devices are running Android 4 or lower, which is not installed by any benign devices in 2020. Hence, we conclude that fraudulent devices run lower Android versions than benign devices do. The potential reasons are two-fold: (i) Using Android phones on lower versions is more cost-effective for attackers to mount a larger scale mobile ad fraud campaign. (ii) The phones with earlier Android versions are much easier to gain full access to the root permission, enabling fraudulent tasks such as auto-clicking with ease. This resonates with the fact that some mobile phone manufacturers (such as Huawei) ban users from unlocking the bootloader on high Android version devices, serving as a requisite for root access acquisition [44].

5 EVILHUNTER

In this section, we present the detailed design and implementation of EvilHunter. The basic insight of EvilHunter is contingent on the cluster-level features rather than any individual device features to identify the fraudulent devices. The main *goal* of EvilHunter is to detect malicious device clusters (click farms) besides identifying fraudulent devices.

In general, EvilHunter is comprised of three stages (see Fig. 4). 1) In the classification stage (Stage 1), based on a series of features discussed in the previous section, EvilHunter designs a device classifier to distinguish fraudulent devices and benign devices by exploiting the features extracted from the ad bid logs; 2) In the clustering stage (Stage 2), EvilHunter proposes a Top-App based Clustering Algorithm, which builds the device graph based on the connectivity features among devices, and then identifies the closely connected device clusters. 3) In the aggregation stage (Stage 3), we classify each cluster by performing majority voting based on the device labels within the cluster and then relabel the devices based on the cluster's classification result, *i.e.*, all devices inside a fraudulent cluster will be labeled as fraudulent. The output of EvilHunter is ([id], label) pairs, indicating which devices are grouped into clusters and whether these clusters are fraudulent or benign.

5.1 Stage 1: Classification

The device classifier stage is a general machine learning classification process. The input is a bunch of ad bid logs while the output is the predicted score s_{dev} for each device, ranging from 0~1. 0 means a high confidential benign score of a device and 1 means a fraudulent one. Device classifier consists of three components: Log-Device Mapper, Device Feature Extractor, and Device Score Predictor.

Module 1.1: Log-Device Mapper. Log-Device Mapper constructs the log-device mapping from the ad bid logs. It then takes the ad bid logs as input and outputs a device-log mapping M, which maps each device ID (id) to the corresponding logs generated by this device. To retrieve the unique id for each device, given an Android device, Log-Device Mapper uses the combination of the MD5 values of IMEI and Android ID. Since both IMEI and Android ID may be an empty value caused by strict permission control enforcement, we use a combination of them to cover more devices in the ecosystem. On the other hand, since Apple restricts the tracking for iOS devices: all iOS apps must have a user's permission to access their IDFAs after iOS 14.5 [7], and therefore the detection of invalid traffic for iOS devices is beyond the scope of this paper.

Module 1.2: Device Feature Extractor. The Device Feature Extractor extracts representative features that can reflect the characteristics

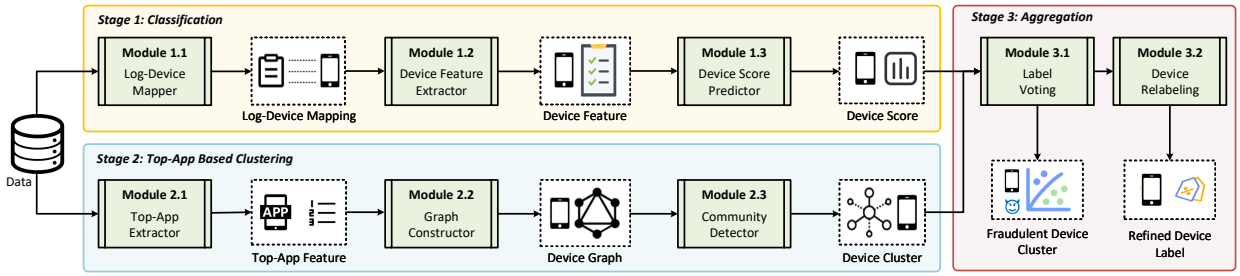


Figure 4: The workflow of EvilHunter.

Table 3: Features extracted by Device Feature Extractor.

Feature Categories	Feature Name
(a) Statistical Features	Number of logs Number of unique IP addresses Number of unique ad slot IDs
(b) Entropy Features	Log entropy IP entropy Ad slot ID entropy
(c) Spatial-Temporal Features	Number of active hours Maximum speed
(d) Inconsistency Features	Number of brands Fake brand ratio Non-browser UA ratio

of fraudulent devices. To achieve this, we defined 11 features that capture the nature of fraudulent devices covering all the fields of the ad bid logs (Table 1), and group these features into four categories (Table 3). These features are extracted as follows.

(a) Statistical features. Device Feature Extractor extracts the statistical features of a device, including the number of log entries, unique IP addresses, and unique ad slot IDs. Intuitively, all these numerical features should be within a certain range, since a normal user cannot use too many different IP addresses or generate too many ad requests.

(b) Entropy features. Device Feature Extractor calculates the entropy of the three features shown in category (b) in Table 3, which measures the uncertainty of the features: higher entropy indicates higher uncertainty. Device Feature Extractor adapts the normalized entropy [29] to compute the features as follows:

$$\eta(X) = -\sum_{i=1}^{n} \frac{p(x_i)\log_2(p(x_i))}{\log_2(N)}, \tag{1}$$

where $x_1, \ldots, x_n$ are n possible results of a feature X (*e.g.*, IP address); $p(x_i)$ is the ratio of x_i in all N logs generated by this device. Device Feature Extractor applies Eqn. 1 on logs, IPs, and ad slot IDs to compute the normalized entropy for them.[1] Note that if $N = 1$, $\eta(X) = 0$.

(c) Spatial-temporal features. Spatial-temporal features are broken down to a given device's active hours and the maximum moving speed. The number of active hours for a device is the total quantity of hours when there was at least one ad bid request sent during that hour. To compute the maximum speed, for each device, Device Feature Extractor uses the `location` and `timestamp` fields to compute the average speed between every two consecutive ad bid requests in the logs and selects the maximum value. To avoid the influence of default location values, we ignore those values including (0,0) and high-frequency locations far away from the target area.

(d) Inconsistency features. Inconsistency features aim to capture the inconsistencies in the logs. The first feature in the category (iv) is the number of device brands for each device. Normally, a device should only have one brand name. So if a device has too many brand names, it may be a signal of fraud. However, the brand fields of a device may be incorrectly reported by app developers in the ad request parameter; to address this issue, Device Feature Extractor extracts another feature, called fake brand ratio, to measure the ratio of fake brands for each device. Device Feature Extractor compares the brand names with two whitelists obtained online,[2] which contain 269 real brand names. If a brand does not appear in any of the two whitelists, Device Feature Extractor considers it as fake. The third feature is the non-browser User-Agent ratio. Normally, the UA field in a log reflects the Browser or *Webview* information of the OS running on the device. The UA is either 'Mozilla' or 'Dalvik'. However, for fraudulent devices, the UA may be forged (*e.g.*, 'Go-http-client'), as it is not a real device. Therefore, Device Feature Extractor uses the non-browser UA ratio as a feature.

Module 1.3: Device Score Predictor. The Device Score Predictor uses traditional machine learning models to perform the training and testing on the features. Particularly, any feature-based classifier (*e.g.*, logistic regression, decision tree, kNN, SVM, and neural network) may potentially be used. However, we note that deep learning is hard to interpret the semantics (or representation) of the features extracted by Device Feature Extractor. In summary, the output of Device Score Predictor is the predicted score s_{dev} for every device.

5.2 Stage 2: Top-App Based Clustering

In Sec. 5.1, we have proposed a novel classifier to distinguish fraudulent devices from benign ones based on the ad bid logs. However, in practice, such *individual* classification results may be affected by the noisy data or the intentional manipulations of the attackers. To address this problem, in Stage 2, we group the devices into various clusters and then exploit the cluster-level features to determine if the clusters and their devices are fraudulent or not. The proposed top-app-based clustering algorithm consists of three steps: extracting top-app features, constructing device graph structure, detecting communities within the graph by applying the *Louvain* algorithm [8] to cluster devices.

[1] To compute the normalized entropy of logs, the logs are first grouped according to the hours of occurrence.

[2] gsmarena.com and kimovil.com.

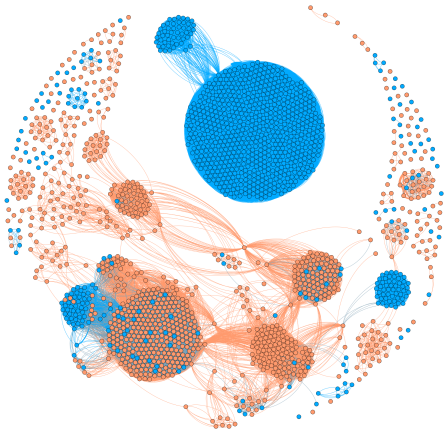


Figure 5: A sample device graph constructed by 1,000 fraudulent vertices (in blue) and 1,000 benign vertices (in orange).

Module 2.1: Top-App Extractor. In this step, we aim at extracting the key features that can represent the *synchronicity* of the devices. Here we choose the app usage patterns as the major factor that we consider, in that the attackers tend to deploy fraudulent devices at a large scale for more revenue. Thus the ad bid requests generated by those fraudulent devices are primarily for the target app. Furthermore, the attackers have to control a huge number of devices automatically, which leads to a similar app usage pattern for those controlled devices. Specifically, the usage pattern UP_{dev} for a device is formulated as:

$$UP_{dev} =< freq(app_1), \ldots, freq(app_n) >, \tag{2}$$

where $freq(app_i)$ denotes the frequency of using app_i in one day.

However, there are thousands of apps in the whole ad ecosystem. Such a feature vector with thousands of elements greatly hinders computation. To reduce the computation complexity, we only keep the values of $freq(app_i)$ for the top η apps in UP_{dev}. We will discuss the detailed parameter selection process in Sec. 6.1 and Appendix A.

Module 2.2: Device Graph Constructor. To construct a device graph using the above features, we define the similarity between a device pair using the cosine similarity as the following:

$$Sim(dev1, dev2) = \frac{UP_{dev1} \cdot UP_{dev2}}{\|UP_{dev1}\| \cdot \|UP_{dev2}\|} \tag{3}$$

Then for each node (device) pair, we add an edge between them and use the similarity as the weight of the edge. To avoid constructing weighted graphs with massive low-weight edges, we set a threshold Sim_{thr} and only add edges between two devices when $Sim(dev1, dev2) \geq Sim_{thr}$.

Module 2.3: Community Detector. An example of the device graph is depicted in Fig. 5. A key observation is that the densely connected clusters are mostly composed of vertices with one type, *i.e.*, either with all fraudulent ones or benign ones. Here we use the popular Louvain method [8, 46] to identify communities in the device graph.

5.3 Stage 3: Aggregation

We aggregate the results of the two stages before Stage 3 via *Label Voting* and *Device Relabeling*.

Module 3.1: Label Voting. First, we use majority voting on the labels obtained from Stage 1 to determine whether the clusters in Stage 2 are fraudulent or benign. Specifically, we compute the average predicted score of the devices for each cluster as the score for the cluster (s_c). If $s_c \geq s_{thr}$, we label the device cluster as fraudulent. Otherwise, the device cluster is benign. Here we only consider the clusters that are composed of more than αN, N is the total number of devices. For example, when $\alpha = 0.1$, we only consider clusters that have more than $0.1N$ devices for label voting.

Module 3.2: Device Relabeling. A significant advantage of Stage 2 is the capability to calibrate the devices labeling taking place in Stage 1. As mentioned in Sec. 5.2, the fraudulent devices may be falsely predicted as benign ones due to the different configurations of the attackers. Thus, for each cluster of size greater than αN, we use the label of this cluster after majority voting to relabel each of its devices.

5.4 Implementation

We implement EvilHunter on a local server equipped with 6 CPU cores, 64 GB memory, and 10 TB SSD, running Windows 10. In Stage 1, the Log-Device Mapper and Device Feature Extractor are implemented using the Scala programming language on the Apache Spark framework. The Device Score Predictor is implemented using Python. In Device Score Predictor, during pre-processing, the features in categories (i), (iii), and (iv) are normalized using the `RobustScaler` in the Scikit-learn package to avoid being stretched by some outliers. We implement 5 classifiers, including Gradient Boosting Decision Tree (GBDT), Multi-Layer Perceptron (MLP), k-nearest neighbors (kNN), Support Vector Machine (SVM), and Logistic Regression (LR). For GBDT, we use LightGBM [22]. We set 20 as the number of early stopping rounds, and enable positive and negative bagging with a bagging frequency of 3. For MLP, we use one hidden layer with 100 neurons. For kNN, the k is set to 15. For SVM, we use LibSVM [9]. For LR, we use the default settings in the *scikit-learn* package [31]. In Stages II and III, we implement all the components via scala on Spark.

6 EVALUATION

In this section, we describe the parameter settings and evaluation results. Specifically, for the stage 1 classifier, we evaluate the detection performance by cross-validation. Secondly, we compare the detection results of EvilHunter with those of Stage 1 alone, to show the performance improvement introduced by Stages II and III.

6.1 Parameter Settings

Table 4 lists all the parameters we use in the system design. To choose the best settings which balance detection accuracy and the computational cost, we start from an initial parameter setting. We then compare the results after tuning each of the parameters. Specifically, we use D_{train} as the training set to train EvilHunter using different parameter settings and use the first day of D_{test} (30k fraudulent devices and 30k benign devices) as the test set to evaluate the performance. The initial parameter settings are ($\eta = 5, Sim_{thr} = 0.5, s_{thr} = 0.5, \alpha = 10^{-4}$). The evaluation is detailed in Appendix A. Based on our evaluation, we choose the optimal parameter settings ($\eta = 5, Sim_{thr} = 0.5, s_{thr} = 0.3, \alpha = 10^{-3}$) in our paper.

Table 4: The parameters in EvilHunter.

Parameter	Explanation
η	The number of top apps considered in UP_{dev}
Sim_{thr}	The minimum similarity to add an edge between two nodes
s_{thr}	The minimum score of labeling a cluster fraudulent
α	The minimum size of a cluster = αN, N is the total device number

Table 5: Cross-validation results across models: mean values and standard deviations (in parentheses).

Model	Accuracy	Precision	Recall	F-score
LightGBM	0.9501 (0.0004)	0.9599 (0.0010)	0.9395 (0.0008)	0.9496 (0.0004)
MLP	0.9496 (0.0006)	0.9603 (0.0015)	0.9379 (0.0019)	0.9490 (0.0005)
kNN	0.9426 (0.0008)	0.9839 (0.0006)	0.9000 (0.0016)	0.9401 (0.0008)
SVM	0.9499 (0.0005)	0.9594 (0.0012)	0.9393 (0.0014)	0.9492 (0.0004)
LR	0.9465 (0.0007)	0.9623 (0.0010)	0.9294 (0.0015)	0.9456 (0.0005)

6.2 Evaluation Results

Dataset. To evaluate the effectiveness of EvilHunter, we randomly select 30k fraudulent devices and 30k benign devices each day from May 6 to May 15, 2020, from D_{2020} as our training dataset D_{train}. After merging the devices with the same IDs, we have 120k/112k unique fraudulent/benign devices. This dataset (D_{train}) serves as a balanced training dataset for the classifier in Stage 1. Using the same method, we obtain 125k/124k unique fraudulent/benign devices from May 16 to May 25, 2020, as our test set D_{test}.

Results of Stage 1. We apply the 5 classifiers as the Device Score Predictor on our dataset to classify the fraudulent devices based on the logs. We follow 5-fold cross-validation. The mean and one standard deviation of the 5-fold test for accuracy, precision, recall, and F score are presented in Table 5. All 5 classifiers achieve over 94% Accuracy, with a Precision rate in excess of 95%. The best model, LightGBM, achieves 95.01% Accuracy, 95.99% Precision, and 93.95% Recall. The classification results have very small variances. We find that there are approximately 3% false positives during the test. The main reason for these false positives is that their statistical features and entropy features are similar to those of fraudulent ones. The users of these devices are probably stimulated by the apps such that they frequently browse or click on the ad contents. The quickly generated ad bid logs of such devices are mostly targeted to the apps where users can earn money by reading news, watching videos, or even viewing ads. As discussed in Sec. 5.2, false negatives in this stage are mainly caused by intentional manipulations of the attackers.

To confirm that the four sets of features are indeed useful for the classification task, we start by using features in the first category presented in Table 3 for classification and adding other feature sets one by one. We use the LightGBM model for classification, and plot the ROC curves for the 4 cases in Fig. 6. As can be seen from Fig. 6, the AUC values keep increasing when more feature sets are used, indicating that all features are effective for the classification task.

Necessity of Stage 2 and Stage 3. To show that it is beneficial to include Stages II and III in the system, we perform an experiment on D_{test}, using 3-stage EvilHunter and State I alone, respectively. Fig. 7 shows the Precision and Recall. It is observed that compared to using State I alone, both Precision and Recall have increased when using 3-stage EvilHunter. Meanwhile, the decrease in Precision is relatively small. This demonstrates that the proposed 3-stage mechanism is more effective and robust in detecting real-world

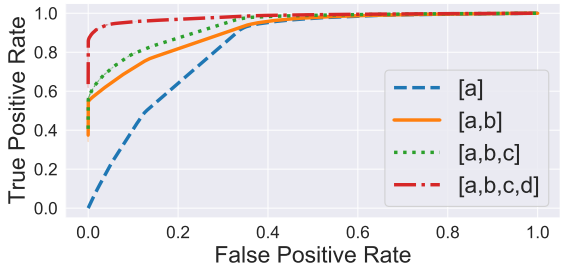


Figure 6: ROC curves; [X,Y] means categories X and Y are used for classification.

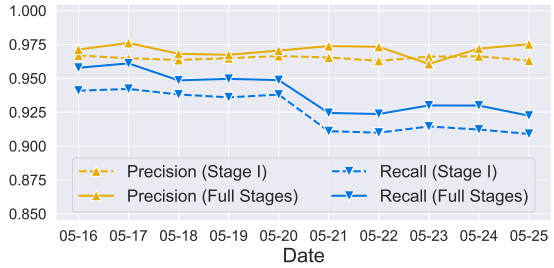


Figure 7: Precision and Recall comparison.

fraudulent devices compared to traditional classifiers. Moreover, as shown in the next section, Stages II and III are important in detecting click farms.

7 CLICK FARM INVESTIGATION

In this section, we give a detailed analysis of the identified click farms (or fraudulent clusters) from D_{test}. By selecting the largest click farm and tracing back to datasets in 2018 (D_{2018}) and 2019 (D_{2019}), we perform an in-depth measurement on the click farm. We introduce our findings and observations, which can help us have a better understanding of how the click farms perform a large-scale and synchronized fraudulent campaign. It will also benefit the community on click farm detection and invalid traffic filtering.

We apply EvilHunter to the first day of D_{test} (May 16, 2020) to identify the click farms, which contains 30k fraudulent devices and 30k benign devices. After the 3-stage process, 176 out of a total of 1069 clusters (with more than 5 devices) are flagged as fraudulent ones. Among 131 clusters consisting of more than 50 devices, 38 clusters are detected as fraudulent.

The largest click farm. We take the largest click farm as an example to show the findings on the fraudulent device clusters, which contains 11,910 devices and 11,910 logs (1 log per device). It is important to point out that many characteristics are not limited to this largest click farm; they also widely exist in other click farms.

7.1 Cheating Strategy 1: Using IP Proxies

It is observed that IP proxy is a widely adopted strategy for the attacker to avoid detection. However, though the attackers can dynamically change the IP address, they may fail to change their geo-location information in some cases. This leads to our two findings: 1) Ad bid logs are located in a small region; 2) GPS and IP geolocations are inconsistent.

Finding 1: The ad bid logs are located in a small region.

Table 6: Statistics of D_{2018} and D_{2019}.

Field	D_{2018}	D_{2019}
Date	Mar 21-Mar 30, 2018	Mar 6-Mar 15, 2019
Log	289,912,853	63,743,968
IMEI	289,850,470	61,113,865
Android ID	100,001	62,774,356
APP	22	9
IP	778,023	249,494
IP subset	8,615	37,834
Location	42	33
Device brand	15	338
Device model	239	1817

There are 1349 devices in this click farm which created 1349 ad bid logs. We find that the GPS coordinates of these ad bid logs can be dramatically gathered into a small area with a 1km radius centered at (xx.64761757174743, xxx.56548085076258). Although the `Location` entry may be forged by tech-savvy attackers, there is no motivation for them to manually set the coordinate to such a specific area. Thus, we speculate that the fraudulent devices in the click farm are physically located there. This offers us a chance to trace back and measure the historical activities of this click farm.

D_{2018} and D_{2019}. We obtain two datasets spanning over 10 days in 2018 and 2019 respectively, and identify the devices and logs whose GPS coordinates are associated with this specific region. The two datasets are described in Table 6. We denote the two datasets as D_{2018} and D_{2019}. The two datasets are used throughout this section.

Finding 2: GPS and IP geolocations are inconsistent.

Based on Finding 1, we further compare the GPS coordinates with the IP geo-localization of the devices in this click farm. We check the distances between the GPS coordinates (LOC) recorded in the logs and the locations corresponding to the IPs (LOC_{IP}) recorded in the logs. First, we use a commercial IP-location query API [6] to fetch the GPS coordinates of the IPs LOC_{IP} for each log. Then, we compute the distance between LOC and LOC_{IP} for each log. The distance distributions of the fraudulent cluster and a benign cluster are shown in Fig. 8. It is clear that there is a huge gap between benign clusters and fraudulent clusters in terms of distance distributions. In the benign clusters, the distances between GPS coordinates and IP locations are less than 40 km. On the contrary, over 75% of logs in the click farms have a distance greater than 400 km. This finding will benefit the ad traffic verification industry by exploiting this feature to identify fraudulent devices.

7.2 Cheating Strategy 2: Rotating IPs and Forging Device IDs

IP/ID filtering is a widely adopted approach in the ad verification industry. Changing IP/ID is a widely adopted strategy by attackers to bypass the detection since traffic verification third parties will pay more attention to the IPs with inflated traffic.

Finding 3: There is a sophisticated strategy of rotating IP addresses and forging device IDs.

IP also serves as a physical fingerprint of devices in many fraud detection methods [29]. Unlike isolated fraudulent devices heavily relying on IP proxies to change IP, the devices in click farms choose a more sophisticated strategy to prevent their invalid traffic from being identified.

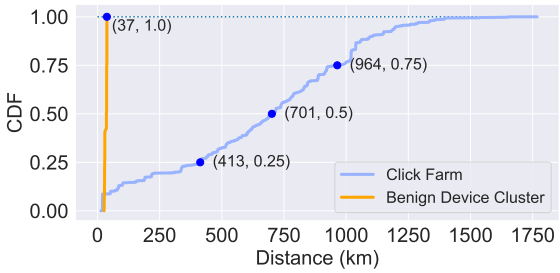


Figure 8: Distance CDF between LOC and LOC_{IP} for a benign cluster and the cluster of case study.

```
ANDROID_VER_CODE = ["4.4.2", "5.0.1", "5.1.1", "6.0", "7.0", "7.1.1"]
DEV_BRAND = ["OPPO", "MST", "CUI", "SAMSUNG", "YUS", "ZTE", "HTC", "UMESI",
             "DAXIAN", "XIAOMI", "VIVO", "YTSP", "MEIZHU", "HUAWEI"]
DEV_TYPE = ["M56", "PLUS5", "Y11", "L1", "M7", "N9", "325p", "MS16", "PLUS 6",
            "F10", "N11", "NOTE 3", "8US", "TUIP95", "MTS 6", "S672", "P8"]
BUILD_CODE = ["KTU84P", "JOP40D", "LMY47X", "LMY48B", "JZO54K", "JDQ39",
              "KOT49H", "LRX21V", "JLS36C"]
UA = f"Mozilla/5.0 (Linux; Android {ANDROID_VER_CODE[randint() % 6]}; " \
     f"{DEV_BRAND[randint() % 14]} {DEV_TYPE[randint() % 17]} " \
     f"Build/{BUILD_CODE[randint() % 9]}) " \
     f"AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/30.0.0.0 " \
     f"Mobile Safari/537.36"
```

Figure 9: User-Agent generation code. The *Chrome WebView* version (in red box) is a fixed value while other variant fields (red lines above) can be generated arbitrarily.

The first strategy is to inflate the IP numbers by randomly changing the IP addresses within the subnet. Specifically, both D_{2018} and D_{2019} are associated with a large number of different IP addresses. However, when zooming in these IP addresses, it is observed that these massive numbers of IP addresses belong to a limited number of subnets. For example, for D_{2018}, it contains 778,023 different IPs, corresponding to only 8,615 subnets, indicating that each subnet contributed to roughly 100 IPs on average. For D_{2019}, there are 249,494 IPs belonging to 37,834 subsets, showing that the IP addresses were scattered across subnets to bypass traffic verification. However, frequently changing IP will make it highly suspicious for the generated traffic as well as the corresponding device IDs. This motivates the click farm to adopt the *second strategy: forging enormous device IDs to reduce the average IP number per device ID.* It is observed that the average IP number per device ID in D_{2018} is less than 1. We also observe that massive devices within the click farm only generate a single ad bid request. The combination of the two strategies helps the attackers evade traffic blocking based on IPs and device IDs.

The aforementioned discussion also largely demonstrates the vast IP resources available to the click farm operators. Furthermore, we note that it is unreliable to verify the invalid traffic exclusively based on IP distributions as recommended by the recent work [29].

Finding 4: The Android IDs have a common prefix.

As discussed before, changing Android IDs is also a strategy widely adopted by the click farm. This is because: (i) any Android ID with too many ad bid logs is generally believed to be suspicious; (ii) changing Android IDs can make the blocking list based invalid traffic filtering approach fail to work. This strategy has also been applied to the identified click farm cluster, which involves 100, 001 Android IDs in D_{2018}.

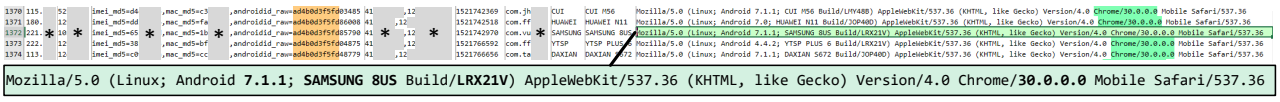


Figure 10: An example of the UA generated by this click farm.

Surprisingly, we find that in 2018, 100,000 out of 100,001 Android IDs show a common pattern: they are the MD5 values of a determinate string $s = s_1 \| s_2$, where s_1 = "ad4b0d3f5fd" and s_2 ="00000",. . . , "99999"; the only exception is ID_x = "ad4b0d3f5fdacd9b". It is derived that ID_x is the Android ID of a real device, and all other Android IDs were created by changing the last 5 digits of it and computing the MD5 values. We confirmed this by looking into the locations recorded in the logs, which showed that these devices were in close proximity to each other. After discussing with Company A, one possible explanation is that the attackers intentionally used this unique Android ID prefix in 2018 as a piece of evidence to request payment from the fraudulent publisher who endorses them to launch the ad fraud campaign. However, in 2019, the Android IDs did not exhibit such a pattern, indicating that the strategies adopted by the attackers have evolved.

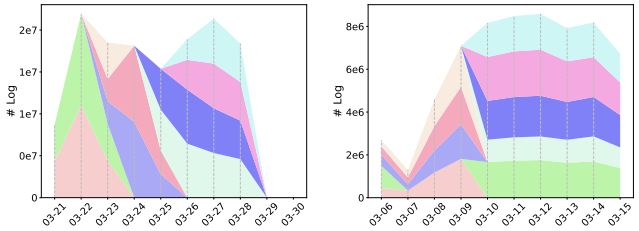
7.3 Cheating Strategy 3: Forging User Agents

User-Agent also includes much critical end-device system information (*e.g.*, Android version, device brand, build code, browser kernel version), which can be exploited to fingerprint the fraudulent devices. This observation drives the click farm devices to generate fake UA fields to conceal their real system information through modifying system configuration files, "`/system/build.prop`" [43], and thus bypass the detection. As the first step towards understanding how to generate the fake UAs, we try to simulate fake User-Agent generation in Python to enumerate all possible combinations of UAs, as shown in Fig. 9. The attackers select items from predefined small sets for the 4 fields (Android version, device brand, device type, and build code [2]) to conceal their critical system information and avoid being blocked.

Finding 5: 3 types of UA fraud are identified.

We then perform a comprehensive analysis of the UA fields of fraudulent devices in D_{2018} and D_{2019}. We have detected the following three UA frauds due to inconsistencies of the fields, which can help us combat fraudulent devices in the future. In the following, we use a real-world UA as an example, which was generated at 2:22:50 AM on March 23, 2018. As shown in Fig. 10, the Android version is 7.1.1, the device brand is "SAMSUNG", the device type is "8US", and the build code is "LRX21V".

1) Chrome WebView vs. Android version. Chrome WebView is an embedded edition of Chrome browser as a non-degradable system component in each version of the Android system. The version (30.0.0.0) was born with Android 4.4 [1] and the updated WebView shipped with Android 4.4.3 has version number 33.0.0.0. It is indicated in [1] that WebView will auto-update for mobile devices with Android L (Android 5.0) and above. In other words, a new Android version with an old Chrome WebView version is highly suspicious. This serves as a strong indicator to identify the fraudulent devices by checking the inconsistency of UA and Android versions. In the example of click farm UA, Android 7.1.1 corresponds to WebView version 52.0.2743.100, which is inconsistent with the old WebView version 30.0.0.0 (Fig. 10).



(a) 2018 **(b) 2019**

Figure 11: Log number proportions of the apps in 2018 and 2019. For each day, the height of the dashed lines separated by different colors represents the log numbers belonging to different apps. There are no overlapping apps between two years.

2) Device brand vs. device type. The second observation is that the device brand and type fields in the click farm UAs are randomly generated, which leads to non-existent device brand/type pairs, such as "SAMSUNG 8US" in the example click farm UA showed in Fig. 10. Importantly, we ascertain that, in the identified click farms, most of the UA fields are evidenced to be forged, such as "HTC M56", "DAXIAN Y11", "UMESI 325p".

3) Android version vs. build code. Android version and build code are in a one-to-many relationship, which can be used to check the forged UAs. In the UA example above, "LRX21V" is the build code for "android-5.0.0_r7" according to the list [2], which falsely combines with the Android version of 7.1.1.

In summary, UA provides a plethora of useful information to determine the nature of a device to be either benign or fraudulent.

7.4 Cheating Strategy 4: Rotating Apps

To avoid massive ad bid requests targeting only one app and being banned by the ad platform, the fraudsters have developed dozens of apps and dynamically changed the apps to run the ad campaign, which will provide economic incentives for the fraudsters.

Finding 6: The apps are uniformly rotated after 3~4 days.

We investigate the proportion of log numbers for 18 apps involved (9 selected apps in 2018 and all the 9 apps in 2019) and plot their transition flows, which are shown in Fig. 11a and Fig. 11b. It is observed that click farms do not generate invalid traffic for a fixed app. Instead, ad spam campaigns have been intentionally launched on different apps from one to another. It regularly takes a certain period to run the ad spam campaign (or rotating period) for each app. Averagely, the length of the period is about 3~5 days. For example, 2 apps were activated on March 21, 2018, and deactivated 3 days later, meanwhile, 3 new apps were activated on March 23, 2018. The remaining 13 apps in 2018 and all the 9 apps in 2019 show the same characteristics. We checked the Android app market

and, interestingly, all of these apps were developed by the common developer. After our manual check, all of these apps do not have any special or practical functionality. It is reasonable to assume that the only purpose of these apps is to run the ad spam campaign. This is also supported by another fact that these apps have not received any updates since being published on the market.

We also apply EvilHunter to the full datasets containing D_{2018}, to investigate how invalid traffic impacts top 50 apps in the 10-day dataset of 2018 (Appendix C).

8 REAL-WORLD DEPLOYMENT

In this section, we will discuss how to implement EvilHunter in real-world applications. As we mentioned in the introduction, in the industry, the major challenge comes from how to address the massive data (*e.g.*, over 50M devices and 100M logs per day) in a reasonable period (*e.g.*, less than 1 day for daily clearance). We propose several techniques to speed up the execution of EvilHunter, in order to support the verification of billions of ad bid requests in an industrial environment. After applying optimization, we apply EvilHunter to an in-the-wild dataset D_{2021} and compare the result with state-of-the-art industry methods. The acknowledgments from our industrial collaborator show that EvilHunter can complement existing industry methods, and is practical to be used in real-world scenarios.

8.1 System Optimization

In the traffic verification industry, it is required to process and detect invalid traffic on a daily basis. Thus, the time cost of processing daily data should be within several hours. In the implementation of Stage 2, a naive method is to compute similarities between each pair of devices and then construct a complete graph. However, such a pair-wise comparison suffers from the scalability issue. Considering hundreds of millions of active devices in the advertising system every day, if we are going to apply the naive method, we need to perform $O(10^{15})$ computation to construct the whole graph, which is unacceptable in practice.

8.1.1 **Optimization techniques.** In the following, we propose three optimization techniques corresponding to the three steps in stage 2 to address the computation challenges.

1) Merging devices. In Module 2.1 Top-App Extractor, since all devices are represented by their top-app features, we can naturally put the focus on distinct features instead of distinct devices. It is observed that in D_{2020}, only a very small proportion of devices (0.25%) have different labels but share the same top-app features. Meanwhile, the number of distinct features is much smaller than the number of devices (<0.5%). To utilize this characteristic, we merge devices with the same feature into a single *vertex*.

For example, suppose two devices `id=bed4f2...` and `id=beed4c...` have the same top-app features: {'e16f25bd' : 1} (*i.e.*, one ad bid log for app id e16f25bd); then we combine the two devices and other devices with the same feature as one single *vertex*, indexed by a unique vertex id. In this way, the vertex number in our graph is significantly reduced, while it does not impact the correctness of the final result. As in D_{2021}, there are 53M devices, which can be merged into 0.25M vertices. The compression rate is over 200x. The pairwise similarity computational time complexity can be decreased from $O(10^{15})$ to $O(10^{10})$.

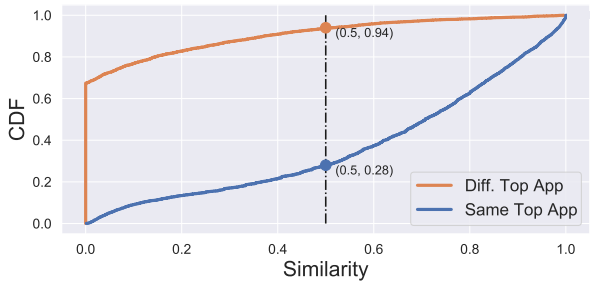


Figure 12: The CDF of pair-wise device similarities.

2) Pruning in similarity computation. While merging devices can decrease the number of vertices to compute, we also find in Module 2.2 Device Graph Constructor that many pair-wise similarity computations are not necessary. Since we only build edges between vertices with high similarities (≥ 0.5), we can omit many similarity computations if the estimated similarity is small (*i.e.*, < 0.5). Recall that we use cosine similarity in our computation. If two vertices have different top-1 apps, then the cosine similarity is very likely to be lower than 0.5. In that case, it is unnecessary to compute the similarity, since it is too small to add an edge between them.

Fig. 12 shows the distribution of similarities between randomly sampled 10,000 device pairs, half of which have different top-1 apps while the other half have identical top-1 apps. The curve in orange shows that there are over 94% of the vertex pairs with different top-1 apps whose similarities are less than 0.5. The computation of such similarities is unnecessary for the subsequent steps. Thus, we only perform similarity computations between two vertices if they share the same top-1 app. In practice, this pruning method can eliminate over 99.7% of the operation each day. Note that this pruning process may influence the detection results since it will result in different device graphs.

3) Dividing graphs for parallel computing. Unfortunately, the complexity of the community detection algorithm (Louvain method) in Module 3.1 Community Detector is $O(nlog_2n)$, which is not efficient enough for large-scale applications. Therefore, we need to find ways to speed up the computation. The input of Community Detector is all the vertices and edges. It is observed that in a large graph, the results of disconnected sub-graphs do not influence each other. We make use of this and divide the problem into sub-problems, and solve them in parallel to increase efficiency. After pruning in similarity computation, the result is a collection of graphs. First, we separate the graphs into groups that have the same top-1 app: each group contains multiple graphs, and all the nodes in graphs of the same group have the same top-1 app. Then, we process the groups in parallel, since there is no dependency among them. Finally, we collect the community detection results from all the groups. This helps us to reduce the processing time from over 48 hours to 40 minutes in daily data processing.

8.1.2 **Evaluation after optimization.** We study how the optimization improves the performance of EvilHunter, and how it impacts the correctness of EvilHunter.

1) Performance improvement. We evaluate the performance of EvilHunter prior to and post-optimization using D_{2021}, which contains 1-day data (53M devices and 117M logs) in 2021. Stage 1 takes roughly 10 minutes; Stage 3 takes 30 seconds. For Stage 2, Module 2.1 Top-App Extractor takes 4 minutes. For Module 2.2 Device Graph Constructor and Module 2.3 Community Detector, before optimization, each of them cannot terminate within 48 hours, respectively; after optimization, they are able to produce results after 48 minutes and 40 minutes, respectively. Therefore, EvilHunter can process 1-day's data within 2 hours after optimization, which meets the practical requirement for daily execution. The performance speedup is more than 28x.

2) Correctness. We also evaluate the potential impact of the optimization with regard to correctness. We randomly select 1,000 vertices (including 21,191 devices) from D_{2021} and apply EvilHunter prior to and post-optimization. The final detection results show that there are only 48 devices (0.22%) with different labels. Therefore, the optimization steps increase the system efficiency by more than 28x, while incurring little loss (0.22%) on the potential correctness, which is acceptable.

8.2 Result Validation

By following the IVT Detection and Filtration Standard [24], industry leaders develop their own detection systems to detect several known types of invalid traffic. The different types of invalid traffic are labeled by different fraud reason codes. To date, there is no specific fraud reason code for detecting click farms, which is the major advantage of EvilHunter. We deploy EvilHunter on 1-day's real-world ad traffic data in order to test the practicality of EvilHunter.

Dataset. We use the ad bid logs in one day (Jan 13, 2021) collected from the real world as our dataset D_{2021}. This dataset contains 53M devices and 117M logs in total.

Results. EvilHunter detects around 8 million (15%) fraudulent devices out of totally 53 million active devices in D_{2021}. These devices generate 37 million (31%) fake bid requests. After Stage 3, there are 23,604 clusters, wherein 5,164 clusters have more than 50 devices, and 491 of them are fraudulent clusters.

Comparison with the detection results of Company A. Since there is no existing industrial system targeting click farm detection, here we use the detection result of the industrial system of Company A, to compare and analyze our result. We find that 93% of the detected invalid traffic by EvilHunter is not detected by Company A's existing detection system. This undetected invalid traffic is originated from click farms. We have manually checked the top 30 click farms containing 5,941,433 devices in total, which contribute to more than 74% of the detected fraudulent devices. We find similar cheating strategies from them as revealed in Sec. 7. Among the 30 click farms, there are 9 click farms that only use IPs from small centralized areas, which contain 329,287 devices (5.5%) in total; there are 14 click farms (3,434,208 devices, 57.8%) rotating IPs and device IDs to evade IP/ID filtering. Note that there are 4 click farms (299,186 devices, 5.0%) that use both two strategies. These 19 click farms are confirmed by Company A, who has adopted our proposed algorithm as a new fraud reason code in their platform.

We also investigate the other 11 detected fraudulent device clusters. Among them, there are potentially 5 falsely detected clusters: 1) 4 of them contain only Android TV devices, which exhibit different behaviors from Android phones; 2) another cluster has different UA fields compared to the normal ones. These UAs are from true Android devices but start with specific values (*e.g.*, bundle ID) instead of `Mozilla` or `Dalvik`. So they are flagged by EvilHunter. There remain 6 detected device clusters whose cheating patterns are not so obvious. They are expected to be double-checked by Company A using auxiliary information.

Consensus with other companies. In addition, we have reported the device IDs and IPs of the detected click farms to the blockchain-based consensus system. They are expected to be cross-checked by other companies in the future.

9 DISCUSSION

EvilHunter update. To deploy EvilHunter and ensure that it captures state-of-the-art ad fraud, we need to periodically collect new ground truth data and retrain the Stage 1 classifier of EvilHunter. This can be achieved by integrating EvilHunter with an active learning approach. For example, we can periodically collect the prediction results of new devices and reuse them with two options: 1) For the devices with high prediction confidence, we can directly use them as the training dataset to retrain the classification model. 2) For the other devices with low confidence, we can manually label the devices using auxiliary information or other existing tools. We show a simplified update process using D_{2020} in Appendix B to demonstrate the practicality and effectiveness of updating EvilHunter per week.

Impact of privacy regulations. To enhance privacy and adhere to General Data Protection Regulation (GDPR), more fields of the ad transaction data are expected to be encrypted or removed, rendering the traffic verification to be more challenging due to a lack of necessary data. However, we argue that completely removing all fields of the ad data is less likely to happen in the near future because it challenges the current user profile based Internet ad ecosystem. Therefore, it is desirable to have a more strict data authorization and access control and limit the data to a small number of highly qualified traffic verification companies and their trusted research partners. We also attempt to investigate differentially private programmatic ad auctions and this deserves separate research.

Open-source datasets. We are in the process of negotiation with our industrial collaborator to release the datasets used in the paper. Once approved, we will release the ad bid logs of fraudulent devices to facilitate future research.

Limitations. EvilHunter shows a good performance in detecting fraudulent devices generating invalid traffic, but EvilHunter mainly focuses on performing the traffic analysis towards the ad bid network; there might exist approaches that can *perfectly* mimic the traffic originating from the benign devices, which can be used to evade our detection. However, in order to create such "perfect" invalid traffic, the attackers need to invest a significant amount of resources to make all features undetectable, which makes it hard to make a profit. It is important to note that the effectiveness of EvilHunter can be further improved if we combine it with other

ad fraud mitigation techniques, such as traffic authentication, honeypots, and dynamic fraud testing.

Additionally, EvilHunter can only be used to detect fraudulent Android devices. Detecting fraudulent devices running iOS is hard for two reasons: 1) Apple has restricted the IDFA permission since iOS 14, which makes it harder to uniquely trace an iOS device in the ad ecosystem; 2) the invalid traffic samples originated from iOS devices are relatively small, due to the difficulty of jail-breaking and hijacking iOS devices. We leave a thorough study of invalid traffic from iOS devices as our future work.

10 RELATED WORK

Ad fraud measurement and detection. Over the past few years, click spam has been extensively studied in the context of web advertising, mobile advertising, and search advertising. The research [30] proposed the characterization of one of the largest click fraud botnets. Researchers also proposed several types of design and analysis of click spam threats [10, 11, 16, 39, 41]. To defend against click spam, many approaches have been proposed to avoid or detect click spam in advertising [13, 14, 17, 27, 38, 40, 45]. Springborn et al. [37] leveraged traffic collected from honeypot websites to identify and analyze a new type of ad fraud, called pay-per-view (PPV) networks. They examined the click spam issue as well. However, we highlight that the industry focus, driven by advertising monetization, has been shifted from click spamming to invalid traffic-enabled coordinated attacks.

To the best of our knowledge, there is only one recent study investigating invalid traffic [29]. The researchers designed a confidence score for each domain, based on the IP entropy. The confidence score offered for each app domain is useful for DSP to determine how to treat the upcoming bid requests. However, this method cannot ascertain what session of ad traffic is invalid; neither can be used to measure the ad traffic at a finer granularity. Instead, the methodology developed in this paper is able to identify the sources of invalid traffic (*i.e.*, fraudulent devices).

System-level ad fraud prevention. Some researchers proposed authentication-based methods to eliminate fraudulent activities in advertising. For example, Juels et al. [21] proposed an authentication method to validate benign users. In [15] and [36], researchers used HMAC-based signatures to check ad click fraud. Li et al. [23] used TrustZone to verify ad clicks and display. However, these solutions rely on the client side's ability to detect anomalies and thus have reduced scalability.

To prevent various types of counterfeit inventories across the advertising ecosystem, by boosting transparency in the supply chain, Interactive Advertising Bureau (IAB) Tech Lab launched the authorized digital sellers (*ads.txt*) project [18]. The project is aimed at publishers and distributors to declare who is authorized to sell their inventory. Furthermore, there are several extended versions of *ads.txt*, including *app-ads.txt* [19] and *ads.cert* [20] to extend to more scenarios. Recently, Pastor et al. [28] proposed another extended version, called *ads.chain*, to resolve the limitations of the previous protocols. However, all of those solutions are designed to increase the transparency in the ecosystem, which is orthogonal to invalid traffic detection proposed in this paper.

11 CONCLUSION

In this paper, we first conduct a measurement study on a labeled ad fraud dataset to distinguish the nature of mobile devices either fraudulent or benign through feature engineering. We then propose and develop EvilHunter, the first mobile ad fraud detection system based on ad bid request logs, which can identify fraudulent devices with high accuracy and automatically identify fraudulent clusters. We reveal several cheating strategies adopted by click farms based on the results of EvilHunter. We further deploy optimized EvilHunter on a 1-day's real-world dataset, which demonstrates its practicality. The results and findings developed in this paper have been acknowledged, and the proposed EvilHunter will be integrated into the platform of our industry partner, a leading ad traffic verification company (Company A), to combat the current burgeoning mobile ad fraud.

ACKNOWLEDGEMENTS

We are grateful to the anonymous reviewers for their constructive feedback. We also thank RTBAsia and China Advertising Association for the long term support. The authors affiliated with Shanghai Jiao Tong University were, in part, supported by the National Natural Science Foundation of China under Grants 61972453 and 62132013. Xiaokuan Zhang was supported in part by the NortonLifeLock Research Group Graduate Fellowship. Minhui Xue was, in part, supported by the Australian Research Council (ARC) Discovery Project (DP210102670) and the Research Center for Cyber Security at Tel Aviv University established by the State of Israel, the Prime Minister's Office and Tel Aviv University.

REFERENCES

[1] 2015. WebView for Android. https://developer.chrome.com/multidevice/webview/overview.
[2] 2020. Codenames, Tags, and Build Numbers. https://source.android.com/setup/start/build-numbers. (Accessed on 09/02/2020).
[3] 2020. Integral Ad Science. https://integralads.com/.
[4] 2020. Oracle Data Cloud. https://www.oracle.com/data-cloud/.
[5] 2020. White Ops. https://www.whiteops.com/.
[6] 2021. IP-API.com - Geolocation API. https://ip-api.com/. (Accessed on 01/21/2021).
[7] Apple. 2021. Upcoming AppTrackingTransparency requirements. https://developer.apple.com/news/?id=ecvrtzt2. (Accessed on 04/21/2021).
[8] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment* 2008, 10 (2008), P10008. https://doi.org/10.1088/1742-5468/2008/10/p10008
[9] Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology* (2011). https://doi.org/10.1145/1961189.1961199 Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.
[10] Geumhwan Cho, Junsung Cho, Youngbae Song, Donghyun Choi, and Hyoungshick Kim. 2016. Combating online fraud attacks in mobile-based advertising. *EURASIP Journal on Information Security* 2016 (2016). https://doi.org/10.1186/s13635-015-0027-7
[11] Geumhwan Cho, Junsung Cho, Youngbae Song, and Hyoungshick Kim. 2015. An empirical study of click fraud in mobile advertising networks. In *2015 10th International Conference on Availability, Reliability and Security*. 382–388. https://doi.org/10.1109/ARES.2015.62
[12] CNBC. 2017. Businesses could lose $16.4 billion to online advert fraud in 2017. https://www.cnbc.com/2017/03/15/businesses-could-lose-164-billion-to-online-advert-fraud-in-2017.html. (Accessed on 08/08/2020).
[13] Vacha Dave, Saikat Guha, and Yin Zhang. 2012. Measuring and fingerprinting click-spam in ad networks. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*. 175–186. https://doi.org/10.1145/2342356.2342394
[14] Vacha Dave, Saikat Guha, and Yin Zhang. 2013. ViceROI: Catching click-spam in search ad networks. In *Proceedings of the 2013 ACM SIGSAC conference on*

Computer & Communications Security (CCS '13). 765–776. https://doi.org/10.1145/2508859.2516688
[15] Michael Dietz, Shashi Shekhar, Yuliy Pisetsky, Anhei Shu, and Dan S Wallach. 2011. QUIRE: Lightweight provenance for smart phone operating systems. In *20th USENIX Security Symposium (USENIX Security 11)*.
[16] Google, White Ops. 2018. The Hunt for 3ve, Taking down a major ad fraud operation through industry collaboration. https://services.google.com/fh/files/blogs/3ve_google_whiteops_whitepaper_final_nov_2018.pdf.
[17] Hamed Haddadi. 2010. Fighting Online Click-Fraud Using Bluff Ads. *SIGCOMM Comput. Commun. Rev.* 40, 2 (April 2010), 21–25. https://doi.org/10.1145/1764873.1764877
[18] IAB Tech Lab. 2019. IAB OpenRTB Ads.txt Public Specification Version 1.0.2. https://iabtechlab.com/wp-content/uploads/2019/03/IAB-OpenRTB-Ads.txt-Public-Spec-1.0.2.pdf. (Accessed on 08/30/2020).
[19] IAB Tech Lab. 2019. IAB Tech Lab Authorized Sellers for Apps (app-ads.txt). https://iabtechlab.com/wp-content/uploads/2019/03/app-ads.txt-v1.0-final-.pdf. (Accessed on 08/30/2020).
[20] IAB Tech Lab. 2020. *OpenRTB, Specification Version 3.0.*
[21] Ari Juels, Sid Stamm, and Markus Jakobsson. 2007. Combating click fraud via premium clicks. In *16th USENIX Security Symposium (USENIX Security 07)*.
[22] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17)*. 3149–3157.
[23] Wenhao Li, Haibo Li, Haibo Chen, and Yubin Xia. 2015. AdAttester: Secure online mobile advertisement attestation using TrustZone. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '15)*. https://doi.org/10.1145/2742647.2742676
[24] Media Rating Council. 2020. Invalid Traffic Detection and Filtration Standards Addendum. http://mediaratingcouncil.org/MRC%20Invalid%20Traffic%20Detection%20and%20Filtration.pdf. (Accessed on 07/18/2020).
[25] Xianghang Mi, Xuan Feng, Xiaojing Liao, Baojun Liu, XiaoFeng Wang, Feng Qian, Zhou Li, Sumayah Alrwais, Limin Sun, and Ying Liu. 2019. Resident evil: Understanding residential IP proxy as a dark service. In *2019 IEEE Symposium on Security and Privacy (SP)*. 1185–1201. https://doi.org/10.1109/SP.2019.00011
[26] Mirror. 2017. The bizarre 'click farm' of 10,000 phones that give FAKE 'likes' to our most-loved apps. https://www.mirror.co.uk/news/world-news/bizarre-click-farm-10000-phones-10419403. (Accessed on 08/26/2020).
[27] Richard Oentaryo, Ee-Peng Lim, Michael Finegold, David Lo, Feida Zhu, Clifton Phua, Eng-Yeow Cheu, Ghim-Eng Yap, Kelvin Sim, Minh Nhut Nguyen, Kasun Perera, Bijay Neupane, Mustafa Faisal, Zeyar Aung, Wei Lee Woon, Wei Chen, Dhaval Patel, and Daniel Berrar. 2014. Detecting click fraud in online advertising: A data mining approach. *Journal of Machine Learning Research* 15 (2014), 99–140.
[28] Antonio Pastor, Rubén Cuevas, Ángel Cuevas, and Arturo Azcorra. 2021. Establishing Trust in Online Advertising With Signed Transactions. *IEEE Access* 9 (2021), 2401–2414. https://doi.org/10.1109/ACCESS.2020.3047343
[29] Antonio Pastor, Matti Pärssinen, Patricia Callejo, Pelayo Vallina, Rubén Cuevas, Ángel Cuevas, Mikko Kotila, and Arturo Azcorra. 2019. Nameles: An intelligent system for Real-Time Filtering of Invalid Ad Traffic. In *The World Wide Web Conference (WWW '19)*. 1454–1464.
[30] Paul Pearce, Vacha Dave, Chris Grier, Kirill Levchenko, Saikat Guha, Damon McCoy, Vern Paxson, Stefan Savage, and Geoffrey M Voelker. 2014. Characterizing large-scale click fraud in ZeroAccess. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS '14)*. 141–152. https://doi.org/10.1145/2660267.2660369
[31] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
[32] PPC Protect. 2019. What Is a Click Farm? The Quick Way to Thousands of Likes. https://ppcprotect.com/what-is-a-click-farm/. (Accessed on 08/26/2020).
[33] Research and Markets. 2019. Worldwide Analysis on the Real-time Bidding Market, 2019 to 2024 - Anticipated to Record a CAGR of 32.9% During the Forecast Period. https://www.prnewswire.com/news-releases/worldwide-analysis-on-the-real-time-bidding-market-2019-to-2024---anticipated-to-record-a-cagr-of-32-9-during-the-forecast-period-300811841.html. (Accessed on 07/21/2020).
[34] Neil Rubens, Mehdi Elahi, Masashi Sugiyama, and Dain Kaplan. 2015. *Active Learning in Recommender Systems.* Springer US, Boston, MA, 809–846.
[35] Burr Settles. 2010. Active Learning Literature Survey. *University of Wisconsin, Madison* 52 (07 2010).
[36] Shashi Shekhar, Michael Dietz, and Dan S Wallach. 2012. AdSplit: Separating smartphone advertising from applications. In *21st USENIX Security Symposium (USENIX Security 12)*. 553–567.
[37] Kevin Springborn and Paul Barford. 2013. Impression fraud in on-line advertising via pay-per-view networks. In *22nd USENIX Security Symposium (USENIX Security 13)*. 211–226.
[38] Ori Stitelman, Claudia Perlich, Brian Dalessandro, Rod Hook, Troy Raeder, and Foster Provost. 2013. Using co-visitation networks for detecting large scale online display advertising exchange fraud. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '13)*. 1240–1248. https://doi.org/10.1145/2487575.2488207
[39] Brett Stone-Gross, Ryan Stevens, Apostolis Zarras, Richard Kemmerer, Chris Kruegel, and Giovanni Vigna. 2011. Understanding fraudulent activities in online ad exchanges. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference (IMC '11)*. 279–294. https://doi.org/10.1145/2068816.2068843
[40] Tian Tian, Jun Zhu, Fen Xia, Xin Zhuang, and Tong Zhang. 2015. Crowd fraud detection in internet advertising. In *Proceedings of the 24th International Conference on World Wide Web (WWW '15)*. 1100–1110. https://doi.org/10.1145/2736277.2741136
[41] Elliott Wen, Jiannong Cao, Jiaxing Shen, and Xuefeng Liu. 2018. Fraus: Launching cost-efficient and scalable mobile click fraud has never been so easy. In *2018 IEEE Conference on Communications and Network Security (CNS)*. 1–9. https://doi.org/10.1109/CNS.2018.8433126
[42] White Ops and ANA. 2019. 2018-2019 Bot Baseline: Fraud in Digital Advertising. https://www.ana.net/miccontent/show/id/rr-2019-bot-baseline. (Accessed on 07/29/2020).
[43] Songyang Wu, Wenqi Sun, Xin Liu, and Yong Zhang. 2018. Forensics on Twitter and WeChat using a customised android emulator. In *2018 IEEE 4th International Conference on Computer and Communications (ICCC)*. 602–608. https://doi.org/10.1109/CompComm.2018.8781056
[44] XDA Developers. 2018. Huawei will stop providing bootloader unlocking for all new devices. https://www.xda-developers.com/huawei-stop-providing-bootloader-unlock-codes/. (Accessed on 09/03/2020).
[45] Haitao Xu, Daiping Liu, Aaron Koehl, Haining Wang, and Angelos Stavrou. 2014. Click fraud detection on the advertiser side. In *European Symposium on Research in Computer Security (ESORICS 2014)*. 419–438.
[46] Haizhong Zheng, Minhui Xue, Hao Lu, Shuang Hao, Haojin Zhu, Xiaohui Liang, and Keith W. Ross. 2018. Smoke screener or straight shooter: Detecting elite sybil attacks in user-review social networks. In *Proceedings of 25th Annual Network and Distributed System Security Symposium, NDSS*.

APPENDIX

A SENSITIVITY OF PARAMETERS

We evaluate the sensitivity of parameter settings of EvilHunter (Sec. 6.1, Table 4). The parameters are listed in Table 4. We start from an initial setting ($\eta = 5, Sim_{thr} = 0.5, s_{thr} = 0.5, \alpha = 10^{-4}$) and change one parameter at a time to find the best parameter settings. All the experiments are conducted 5 times, and we report the mean values in this section. Note that the standard deviations are very small (<1%) in all the experiments.

η. We have found that η does not impact the final result too much (in Fig. 13). Meanwhile, the larger η is, the longer the total time cost will be. However, we cannot simply choose the minimum value (1) for η: much information would be discarded if we only keep the top-1 app. Here we define the $loss(k)$ as the ratio of discarded ad bid logs compared to $\eta = \infty$ when setting $\eta = k$. As shown in Fig. 13, there is a tradeoff between *loss* and the time cost when η increases; *loss* is decreased to 0.4% when $\eta = 5$. Therefore, we choose $\eta = 5$ as the optimal setting.

Sim_{thr}. Similar to η, we also find that Sim_{thr} has little effect on the accuracy (Fig. 14). Recall that the main goal of Sim_{thr} is to drop edges with low weight. The value of it has a huge impact on the clustering result: if a large Sim_{thr} (*e.g.*, 0.9) is chosen, many edges with a lower weight will be deleted, which will lower the size of individual clusters. Therefore, we evaluate the number of devices (*num*) in the biggest cluster using different Sim_{thr} to make sure that a reasonable Sim_{thr} is chosen. The result in Fig. 14 shows that

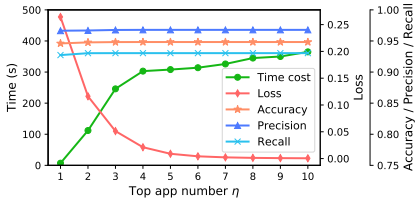


Figure 13: η Experiment

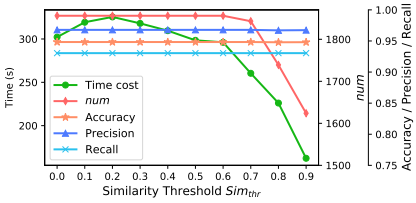


Figure 14: Sim_{thr} Experiment

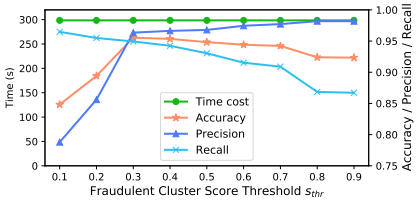


Figure 15: s_{thr} Experiment

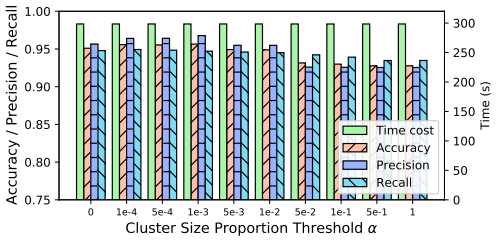


Figure 16: α Experiment

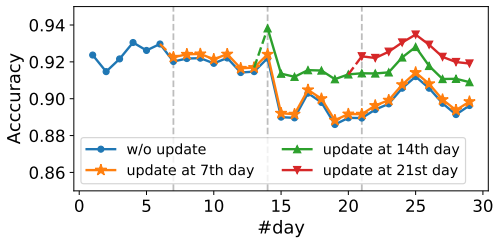


Figure 17: Accuracy with and without updating.

once $Sim_{thr} >= 0.7$, *num* starts to decrease, which means the clustering result is negatively impacted by Sim_{thr}. Moreover, the time cost starts to decrease when $Sim_{thr} >= 0.2$; it plateaus between $Sim_{thr} = 0.5$ and $Sim_{thr} = 0.6$. Based on the above observations, we choose $Sim_{thr} = 0.5$.

***s_{thr}* and *α*.** We use similar methods to choose the best values of s_{thr} and α. Because they are only involved in the aggregation stage (Stage 3), their impact on the time cost is negligible. When s_{thr} increases, the accuracy and precision increase while the recall drops (Fig. 15). $s_{thr} = 0.3$ is the turning point, so we choose $s_{thr} = 0.3$ as our setting. For α (Fig. 16), when α increases, accuracy and precision first increase until α reaches 10^{-3}, then decrease after that. Meanwhile, recall drops slowly all the time. Therefore, we choose $\alpha = 10^{-3}$.

Summary. Based on our evaluation, we use ($\eta = 5, Sim_{thr} = 0.5, s_{thr} = 0.3, \alpha = 10^{-3}$) as the optimal settings, and use the settings in the paper.

B SYSTEM UPDATE

In practice, attackers will keep evolving their cheating strategies to avoid detection. Therefore, EvilHunter must be able to update periodically. In this section, we present a simplified update scheme, which updates EvilHunter per week.

Methodology. In Stage 1, we periodically retrain the classifier using an active learning approach [34, 35]. At the end of each week, we collect the devices labeled by EvilHunter in the last week (7 days) and use them to retrain the classifier if the confidence of the prediction is high, *e.g.*, the predicted score is within $[0, 0.1]$ (for benign devices) or $[0.9, 1.0]$ (for fraudulent devices). Alternatively, new datasets can be obtained using other means (*e.g.*, from other companies) to retrain the classifier. For the threshold parameters used in Stages II and III, we keep these parameters as fixed values for incoming new datasets until the result of offline cross-validation

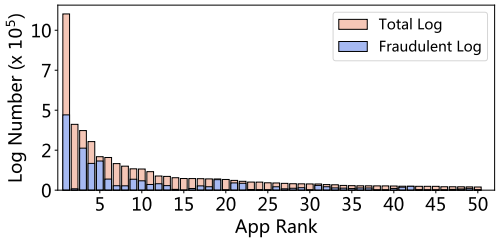


Figure 18: Log number distribution of the top 50 apps.

significantly drops. Once it happens, we search for the threshold parameters as we did in Appendix A.

Evaluation. We use D_{2020} to perform an evaluation of our weekly updating scheme. We compare the accuracy with and without updating in Fig. 17. The first model (M1, in blue) is trained with the data of day 0. The second model (M2, in yellow) is an updated version of M1, retrained using the labels of the first week at the beginning of the second week (7th day). M2 is then tested on the dataset since the 7th day. Similarly, the third model (M3, in green) is trained with the dataset of the first two weeks, and tested using the data of the last two weeks; the fourth model (M4, in red) is trained with the data of the first three weeks and tested using the last week's data. The results suggest that our updating scheme can indeed improve the accuracy.

C PROFILING TOP 50 APPS

To have a deeper insight into the ad fraud caused by invalid traffic in 2018, we use the following method to build suitable versions of EvilHunter in 2018 and profile top 50 apps.

Methodology.

Since fraudulent devices may exhibit different features in different years, we can not directly apply the trained model in 2020 to predict the old devices in 2018. Thus we retrieve the device IDs of the labeled devices in 2020 from the full bid logs of 2018. As a result, we found a total of 3,840 fraudulent and 5,070 benign devices in 2018.

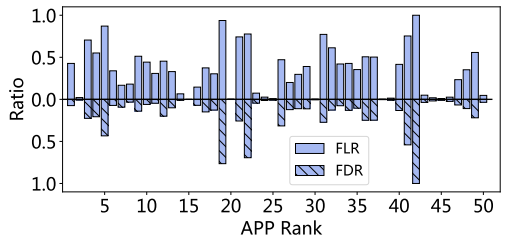


Figure 19: The distribution of fraudulent log ratios (FLR) and fraudulent device ratios (FDR) of the top 50 apps.

Then we use these devices as the training dataset and then train the device classifier. The classifier achieves 80.7% accuracy, 78.5% precision, and 86.1% recall in the 2018 dataset. To study the top apps sending the largest number of ad bid requests, we use 300,000 random sampled Android devices in the 2018 dataset, predict them with EvilHunter, and extract the bundle IDs contained in their logs. We then calculate the number of logs for each app, as well as the number of fraudulent logs, *i.e.*, logs generated by fraudulent devices. We define the following metrics (Eqns. 4–6), including a fraudulent device ratio (FDR), a fraudulent log ratio (FLR, identical to a previous invalid traffic ratio), and an app fraudulent degree (AFD) to represent the fraud degree for each app.

$$\mathrm{FDR_i} = \frac{\#\text{ Fraudulent devices of app } i}{\#\text{ Total devices of app } i}, \tag{4}$$

$$\mathrm{FLR_i} = \frac{\#\text{ Fraudulent logs of app } i}{\#\text{ Total logs of app } i}, \tag{5}$$

$$\mathrm{AFD_i} = \begin{cases} \text{low}, & \mathrm{FLR_i} \in [0, 0.33), \\ \text{medium}, & \mathrm{FLR_i} \in [0.33, 0.66), \\ \text{high}, & \mathrm{FLR_i} \in [0.66, 1]. \end{cases} \tag{6}$$

The results of the top 50 apps, with respect to log numbers, and detailed statistics of the top 5 apps are shown in Fig. 18 and Table 7, respectively. From Fig. 18 and Table 7, we can see that the top apps generated 4,974,027 ad bid requests, and 1,925,669 requests were invalid; the top 5 apps all together generated more than 200,000 requests. We further studied FLR and FDR, as shown in Fig. 19. We can see that 14 out of the top 50 apps exhibited high FLR (>50%), wherein 6 apps had an FDR higher than 30%. This indicates that a relatively small number of fraudulent devices generate a larger number of ad requests.

Table 7: Statistics of the top 50 apps sending most ad bid requests in the 2018 sampled dataset (300,000 devices). FLR is the fraudulent log ratio (invalid traffic ratio); FDR is the fraudulent device ratio; AFD is the app fraud degree (defined in Appendix C). The results are derived from EvilHunter. The daily loss is the estimation based on the average eCPM value ($2). # 19, # 22, and # 48 are iOS apps with unknown numbers of downloads. # 49 is unavailable in any market.

Rank	App Bundle ID	Version	# Downloads	Latest Update	# Logs	FLR	# Devices	FDR	Available	AFD	Daily Loss
1	com.cl**********	6.03.5	5 B	2020/07/31	1,102,014	42.72%	17,455	7.44%	Yes	Medium	$303,461
2	com.xu**********	5.55.2	2 B	2020/07/31	411,661	2.13%	7,929	0.88%	Yes	Low	$5,645
3	com.ij**********	5.3.3	528 M	2019/10/23	372,631	70.51%	785	22.55%	Yes	High	$169,360
4	com.cl**********	4.2.3	760 M	2020/07/11	303,726	55.13%	3,776	20.68%	Yes	High	$107,920
5	com.co**********	5.1.2	211 M	2019/08/09	209,332	87.11%	613	43.39%	Yes	High	$117,528
6	com.co**********	6.6.0	674 M	2020/07/12	204,402	33.93%	1,122	7.22%	Yes	Medium	$44,698
7	com.ji********	2.8.26	3 B	2020/07/29	166,085	16.75%	6,816	9.26%	Yes	Low	$17,929
8	com.an**********	12.7.10	12 B	2020/08/07	150,159	18.12%	768	3.26%	Yes	Low	$17,534
9	net.mo**********	2.6.2	69 M	2020/05/15	133,471	51.25%	50	14.00%	No	Medium	$44,088
10	com.sh*********	2.4.602	534 M	2020/07/31	132,317	44.26%	797	6.02%	Yes	Medium	$37,742
11	com.qi**********	2.6.2	104 M	2020/07/27	115,721	30.84%	2,500	4.64%	Yes	Low	$23,000
12	com.ou**********	2.0.3	32 M	2013/08/16	88,555	45.38%	30	20.00%	No	Medium	$25,904
13	com.so**********	2.0.2	397 M	2019/12/27	86,111	33.00%	1,094	10.05%	Yes	Low	$18,318
14	do*********	7.5.3	367 M	2020/02/13	77,829	6.55%	756	1.19%	Yes	Low	$3,287
15	com.wa*****	3.5.7	14 M	2019/06/04	72,880	0.00%	43	0.00%	No	Low	$0
16	com.ca*********	3.1.2	262 M	2020/07/20	72,562	14.56%	1,751	7.08%	Yes	Low	$6,810
17	com.ma**********	2.4.76	737 M	2020/07/30	71,844	37.42%	777	14.93%	Yes	Medium	$17,326
18	com.tv**********	5.2.3	43 M	2020/05/09	70,479	30.38%	322	12.73%	Yes	Low	$13,799
19	com.mo**********	1.1	-	2020/06/24	70,073	93.74%	68	76.47%	No	High	$42,339
20	com.ne**********	33.1	3 B	2020/07/29	66,937	0.05%	5,417	0.02%	Yes	Low	$19
21	com.zx********	2.4.1	1 M	2017/09/12	61,161	74.25%	171	25.73%	No	High	$29,270
22	com.ne*********	33.1	-	2020/07/22	55,420	77.70%	1,637	69.40%	Yes	High	$27,756
23	com.sh**********	8.6.4	7 B	2020/07/31	50,700	7.38%	8,776	4.49%	Yes	Low	$2,411
24	com.le*******	2.4.6	1 B	2020/07/23	50,392	2.66%	3,291	1.37%	Yes	Low	$862
25	com.yd******	1.0	1 K	2018/01/23	46,092	0.71%	4,906	1.04%	No	Low	$212
26	com.hu**********	1.1.1	1 K	2019/07/26	45,040	47.12%	19	31.58%	No	Medium	$13,678
27	com.ly**********	1.1.8	36 M	2019/12/11	42,581	20.07%	90	12.22%	Yes	Low	$5,507
28	fl**********	1.0	10 K	2017/03/29	40,328	29.72%	955	11.10%	No	Low	$7,725
29	com.wt**********	3.6.2	1 M	2016/01/26	39,662	39.07%	133	11.28%	No	Medium	$9,987
30	com.du**********	3.6.5	1 B	2020/08/04	39,005	0.16%	1,234	0.57%	Yes	Low	$39
31	com.sh*********	2.1.4	3 M	2019/06/15	38,694	77.24%	88	27.27%	No	High	$19,263
32	com.zx********	3.7.702	63 M	2020/05/07	34,975	61.28%	236	12.71%	Yes	Medium	$13,815
33	com.du**********	2.7.5	20 M	2017/06/28	33,671	42.04%	2,599	7.58%	No	Medium	$9,124
34	com.fo**********	5.0.5	17 M	2019/09/17	29,673	42.72%	46	13.04%	Yes	Medium	$8,169
35	com.hu**********	7.0.21	203 M	2020/07/25	28,223	35.41%	544	10.48%	Yes	Medium	$6,442
36	com.pa**********	1.19	1 K	2018/01/19	27,328	50.60%	3,463	24.86%	No	Medium	$8,912
37	com.pa**********	3.9.11	4 K	2018/01/03	27,201	50.27%	3,480	24.77%	No	Medium	$8,813
38	com.bl**********	2.3.6	109 M	2020/08/03	26,869	0.06%	293	0.34%	Yes	Low	$9
39	com.ba********	8.2.0	1 B	2020/08/03	26,689	1.43%	1,513	1.12%	Yes	Low	$245
40	com.po**********	0.9	668 K	2018/11/10	26,004	41.60%	83	13.25%	No	Medium	$6,972
41	com.ba**********	7.13.0	-	2020/07/27	24,767	75.36%	1,399	54.11%	Yes	High	$12,030
42	com.ca**********	3.0.1	52	2017/11/24	24,274	100.00%	2	100.00%	No	High	$15,645
43	com.lg**********	2.0.0	7 M	2019/09/20	24,117	4.96%	720	3.61%	Yes	Low	$771
44	com.ub**********	3.5.2	206 M	2020/06/12	24,108	1.87%	317	1.89%	Yes	Low	$290
45	com.ln******	1.0	1 K	2018/01/12	23,915	0.84%	5,360	1.19%	No	Low	$128
46	com.cl*********	5.1.2	2 M	2019/09/23	21,411	2.61%	703	2.42%	No	Low	$359
47	com.mo**********	1.0.0	1 M	2019/03/11	21,363	23.40%	3,430	6.68%	No	Low	$3,222
48	com.ru********	7.1.3	180 M	2020/08/06	21,148	35.17%	629	10.97%	Yes	Medium	$4,794
49	com.li**********	-	-	-	20,599	55.76%	41	21.95%	-	Medium	$7,403
50	com.an**********	1.0.0	8 M	2018/03/27	19,828	4.71%	773	3.49%	No	Low	$602