MadDroid: Characterizing and Detecting Devious Ad Contents for Android Apps

Anonymous Author(s)

ABSTRACT

Advertisement drives the economy of the mobile app ecosystem. As a key component in the mobile ad business model, mobile ad content has been overlooked by the research community, which poses a number of threats, e.g., propagating malware and undesirable contents. To understand the practice of these devious ad behaviors, we perform a large-scale study on the app contents harvested through automated app testing. In this work, we first provide a comprehensive categorization of devious ad contents, including five kinds of behaviors belonging to two categories: *ad loading content* and *ad clicking content*. Then, we propose *MadDroid*, a framework for automated detection of devious ad contents. *MadDroid* leverages an automated app testing framework with a sophisticated ad view exploration strategy for effectively collecting ad-related network traffic and subsequently extracting ad contents. We then integrate dedicated approaches into the framework to identify devious ad contents. We have applied *MadDroid* to 40,000 Android apps and found that roughly 6% of apps deliver devious ad contents, e.g., distributing malicious apps that cannot be downloaded via traditional app markets. Experiment results indicate that devious ad contents are prevalent, suggesting that our community should invest more effort into the detection and mitigation of devious ads towards building a trustworthy mobile advertising ecosystem.

1 INTRODUCTION

Two-thirds of the world's population is now connected via mobile devices [37], and the share of smartphones among such devices has rapidly increased in recent years. For example, the official Google Play market for Android currently hosts over 3 million apps [14]. Moreover, most apps on markets are free. There is also a trend showing that more and more paid apps have been released as free ones by their developers [71], suggesting that the business model in free apps offers potentially more attractive revenue. In most cases, while users do not pay to install and run the apps, developers can still monetize through displaying advertisements (or ad in short) on app User Interfaces (UI). It is estimated that the size of the global mobile ad market would reach 215 billion US dollars by 2021, which will represent 72% of the total digital budgets [29].

Unfortunately, the mobile ad business model has been abused by malicious individuals to make undue benefits. For example, unscrupulous app developers are attempting to cheat both advertisers and users with fake or unintentional ad clicks so as to earn profits [19, 22, 31, 49]. As revealed by a recent report, mobile advertisers have approximately lost 1.3 billion US dollars due to ad fraud in 2015 alone [32], making research on malicious mobile advertisement a critical endeavor for sanitizing app markets.

Fortunately, the research community becomes increasingly interested in this area with a variety of research directions targeting the ecosystem of mobile ads. For example, researchers have investigated topics such as automated detection of ad networks [15, 42, 44, 51], security and privacy analysis of ad libraries [24, 35, 59], and the detection of mobile ad frauds [19, 22, 31, 49]. Nevertheless, these studies have so far targeted mobile ad issues from the perspectives of either *app developers* or *ad networks*. The latter plays the role of trusted intermediary platforms for connecting *mobile advertisers* to *app developers* by providing toolkits (e.g., ad SDKs) to be embedded in apps. The perspective of *mobile advertisers* themselves, who provide *ad contents* and pay ad networks, has been rarely studied.

Despite being a key component in the mobile ad business model, mobile *ad content* has been overlooked by the research community. Yet, ad content poses a number of threats. On one hand, ad content downloaded at runtime from trusted ad networks could serve as a channel for attackers to distribute undesirable contents or even malware. For example, even Google Play apps have been reported to display porn ads [2, 3]. Recent reports also suggested that some ad contents actually come with the CoinMiner malicious script which uses the device's physical resources in the background to mine digital currency [4]. On the other hand, besides the ad content itself, some unwanted payload may be triggered when the user interacts with the ad content. For instance, the ad clicking event could redirect the current execution page to a malicious website. Overall, we refer to such ad contents as ***devious***, since they are deceitful for all parties (i.e., for app users, for app developers, and potentially for ad networks when they are unaware of this bad behavior of mobile advertisers).

To the best of our knowledge, there lacks an in-depth study on both *ad loading contents* and *ad clicking contents*. The closest studies including Chen *et al.* [16] and Shao *et al.* [62] only examine ad clicking contents, and thus have several limitations (detailed in the evaluation section) and overlook numerous ad clicking contents. In this paper, we fill this gap by performing a comprehensive study of mobile ad contents, aiming to understand the state of practice in devious ad contents and devise practical techniques for preventing their spread in the mobile ecosystem. To this end, we first present a systematic approach to categorizing devious mobile ad contents based on a thorough investigation of ad-related policies and reports (**Section 3**). Then we design and implement *MadDroid*, a prototype framework for automated detection of devious mobile ad contents (**Section 4**). *MadDroid* leverages a dedicated automated app testing approach to explore ad views in an app, based on a sophisticated ad-first strategy (**Section 4.1**). While exploring mobile ads, *MadDroid* records any network traffic and collects contents exchanged between mobile ad networks, advertisers and user devices. By hooking the HTTP-related APIs in the Android framework, *MadDroid* manages to precisely locate ad traffic from all recorded traffic. (**Section 4.2**). Finally, we implement in *MadDroid* a number of specialized approaches (**Section 4.3**) to detect specific types of devious ad contents using techniques such as deep learning, optical character recognition (OCR), etc.

To summarize, we make the following main contributions:

- **A novel ad traffic identification approach**. We present a HTTP hooking approach (by hooking the HTTP-related APIs) to *iteratively* build a mapping between ad libraries and ad hosts. This mapping enables our approach to precisely pick out ad traffic from general network traffic. The detailed experiment results suggest that, our approach outperforms state-of-the-art ad traffic identification methods significantly, i.e., we have identified three times of the ad hosts and increased the collection of ad contents by 126%.
- **A comprehensive detection framework.** We propose *MadDroid*, a framework to detect devious mobile ad contents. To the best of our knowledge, this is the first attempt in the literature to detect five groups of devious mobile ad contents.
- **A large-scale study in the wild.** We conduct a large-scale empirical evaluation on the usefulness and effectiveness of *MadDroid*. By applying *MadDroid* to 40,000 apps, we find roughly 6% of apps (2,322) that deliver devious ad contents. We will release *MadDroid*, along with all the experiment results to the research community, to further boost the research on this direction.

2 BACKGROUND AND TERMINOLOGY

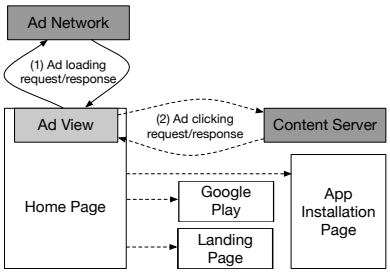

Figure 1: The general working process of mobile ads.

In order to clarify the meaning of specific terms used in this paper, and to help readers get an overall understanding of how mobile ads work, we briefly describe the workflow of mobile ad delivery on users' device interfaces. Fig. 1 illustrates the overall workflow. For simplicity, we will refer to any graphical user interface where an ad can be displayed as a **Home Page**. When such a page appears on the foreground of a device's screen (e.g., after a menu item is selected), an ad-related HTTP request is sent in an attempt to fetch *ad content* from an **Ad Network**. In the mobile ecosystem, the Ad Network plays the role of a trusted intermediary platform that connects **advertisers** to **app developers** by providing ad libraries (e.g., Google's AdMob) to be embedded in app code for fetching and displaying ads at runtime. In response to the ad-related HTTP request, the Ad Network may serve for example an image that will be used on the Home Page to update an ad view.

Once the ad view is displayed on the Home Page, users can click it to observe its content. Normally, when the ad is clicked, it will again trigger another ad-related HTTP request that attempts to fetch additional *ad contents* from a **Content Server**, which may be hosted by advertisers or other third-parties. There are three types of ad contents that Content Servers recurrently push to users:

(1) A redirection link that switches the current Home Page to a so-called **Landing Page** for displaying the ad information, such as an online shopping page where the user can purchase the items that were usually advertised on the Home Page.
(2) A deep-link that switches the current Home Page to **Google Play** for helping users install advertised apps.
(3) Automatic download of a file. Typically, this is an APK file. When the APK downloading is completed, the current Home Page is switched to an **App Installation Page**, where users can decide whether to install the downloaded app.

As shown in Fig. 1, there are two types of ad-related HTTP requests: one as *Ad loading* request and the other as *Ad clicking* request. Unfortunately, the ad content served in response to both requests may be comprised of devious artifacts that may threaten the security and privacy of app users.

3 MOTIVATION AND CATEGORIZATION

We first describe a real-world example of devious ad contents that we have encountered on a popular racing game app. We then create a categorization of devious mobile ad contents based on a thorough investigation of ad-related policies and reports, which will drive the implementation of techniques for identifying devious ad contents.

3.1 Motivating Example

While manipulating the free racing game app *Speed Racing Ultimate*[1], it is not uncommon to see ads appearing on the foreground. Fig. 2 provides the screenshot of an example ad view observed by one of the authors while playing the game. At the top right corner, there is a *cross* symbol (×), which conventionally suggests that the ad view can be closed by clicking at this location. Once clicked, however, a redirection is triggered and the current home page is replaced by a landing page where ad content is displayed. At first, one may suspect that the user failed to properly click on the correct location, instead clicked on the actual ad, justifying the behavior. Nevertheless, after several failed attempts, the user concludes that the "close" functionality is not supported, or at least not working as expected, via the cross symbol. Further manual investigations into the ad later revealed that *the (×) symbol is actually embedded in the image*. This demonstrates a deceitful behavior as the purpose of the cross symbol was never to close the ad but to trick users into clicking on the ad. Such devious ads are increasingly frequent in practice, however, studies about them are scarce in the literature.

Figure 2: An example of click-deceptive Image.

3.2 Categorization of Devious Ad Contents

The consequence of the redirection triggered by the devious ad content example presented above was a simple annoyance for users.

[1] com.minicliphr.speedracingultimatefree [5]

However, we can imagine scenarios where such a redirection lands on malicious payload being performed. Thus, motivated by such possibilities, we decided to conduct a systematic study of the current devious mobile ad contents. To categorize such contents, we first investigate the undesirable mobile ad contents from: (1) the policies related to mobile ad contents of popular app markets [6, 33, 53, 54, 70**?**], (2) media reports in news outlets [2–4, 25, 38, 40, 69], and (3) some real-world apps that host devious ad contents. Based on our empirical investigation, we summarize the observed devious ad contents into five (5) groups enumerated in Fig. 3.

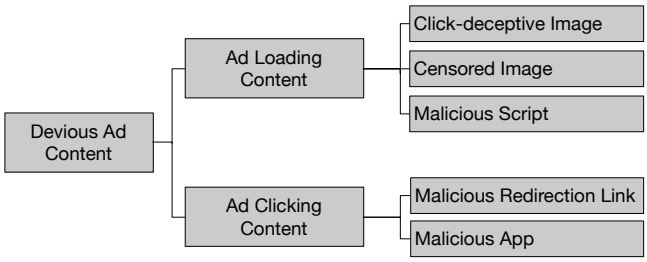

Figure 3: The five categorized groups of devious ad contents.

Note that three groups, namely *Click-deceptive Image*, *Censored Image*, and *Malicious Script*, are related to ad content obtained following the Ad Loading request, while the remaining two, namely *Malicious Redirection Link* and *Malicious App* are related to ad contents obtained after an Ad Clicking request.

We then describe each group in detail.

(1) Click-deceptive Image: As shown in Section 3.1, devious ad networks (or advertisers) may provide, as ad content, a click-deceptive image, where a "cross" symbol (×, or similar images) is directly embedded in the ad image aiming at tricking users into clicking on it to close the ad view. Normally, the "close" button of an ad is displayed as a separate image: on one hand, this allows ad networks (or app developers) to set events (such as click to close) that are independent from events associated to the ad image (such as click to follow a link); on the other hand, setting a separate image offers the opportunity to display the "close" button after a delay of several seconds, giving enough time for users to notice the ad.

(2) Censored Image: Censored Image refers to such ad images that fall under censorship with respect to state legislation or the market policy. In this work, we enumerate the cases of *Gambling*, *Violence*, *Medical*, and *Pornographic* images, which might be prohibited. Google itself explicitly warns developers that gambling advertising should abide by local gambling laws and industry standards [6]. Similarly, Google also disallows the presentation of violent ads as they are not appropriate for children, while some medical-related contents cannot be advertised at all [33]. Adult ads also need to comply with certain policies: for example, it is not allowed to distribute ad contents that may be interpreted as promoting a sexual act in exchange for compensation in many countries. Besides Google Play, many third-party app markets [53, 54, 70] do not allow advertising of Gambling and Pornographic contents.

(3) Malicious Script: Mobile ads, which are usually displayed via the WebView widget in Android, can legitimately run code to interact with the host app. For example, a code fragment can be included to remove the ad after the close button is clicked. Unfortunately, devious ad networks may inject malicious scripts in the ad. For example, the 360 Fenghuo Lab, has reported that some ad networks distribute devious ad contents through which they mine bitcoins on users' devices, without their knowledge [4].

(4) Malicious Redirection Link: Some mobile ads, after clicked, may jump to landing pages where malicious contents are presented to the users. When such redirected content is clicked, the security and privacy of the user may be in jeopardy.

(5) Malicious App: This group refers to such mobile ads that, when clicked, may download malicious Android apps into the user device. In this scenario, devious ad contents appear as an attractive means to distribute malware on user devices.

3.3 Challenges

In this work, we aim at proposing an effective approach to detect these aforementioned types of devious ad contents from Android apps. It is nevertheless non-trivial to achieve this automatically. There are at least three challenges that need to be effectively addressed. The three challenges are summarized as follows.

- **How to automatically trigger and collect ad content?** Mobile ad contents could be collected at the time when the ad is fully loaded or consumed, which requires not only triggering the appearance of mobile ads but also clicking the presented ads. Unfortunately, mobile ads could be delivered in different sizes (e.g. Banners, Interstitials, Full Screens), different carrier widgets (e.g. WebView, ImageView, ViewFlipper), different numbers and places (one or multiple, within the same UI state or different states), sophisticated approaches hence are needed to effectively traverse ads in apps while ensuring good coverage.
- **How to efficiently pick out ad traffic from general network traffic?** Mobile ad contents can be extracted from the network traffic, specifically the ad-related traffic (or ad traffic in short). However, when collecting ad traffic at app running time, general network traffic (e.g., download a file) would be also collected, i.e., non-ad traffic and ad traffic are inevitably mixed. Hence, there is a need to design effective approaches to separate ad traffic from the general ones.
- **How to precisely differentiate devious ad contents from normal ad contents?** With a systematic approach, we have identified and categorized five groups of devious mobile ad contents, which respectively need specialized techniques to characterize. Considering that new groups of devious ad contents can be added in the future, the detection approach should not only be inclusive (e.g., cover all the devious groups), but also extensible (e.g., can be easily extended to cover new devious groups).

4 APPROACH

Fig. 4 depicts the essential modules of the workflow in our proposed *MadDroid* framework. Towards detecting devious ad contents which are delivered to an input Android app, we propose an architecture with three modules that respectively address the aforementioned three challenges:

Figure 4: Overview of the *MadDroid* framework.

- **TCM**: a network Traffic Collection Module, which focuses on traffic generated as part of the ad loading or interaction phases. This module requires careful design as it requires dynamic execution which, in order to be effective, must be focused on covering mainly ad-involved UIs.
- **CEM**: an ad Content Extraction Module, which carefully learns to identify, among exchanged traffic, which ones are about loading content that must be extracted. This extraction further explores contents that are delivered after an ad view is clicked.
- **DDM**: an ad Deviousness Detection Module, which finally analyzes the extracted ad contents to identify devious ones. Given the diversity of devious ad contents, this module implements specialized detection schemes with adapted techniques ranging from character recognition to deep learning.

This modularized architecture enables flexibility for extension and maintenance. Given that the categorization presented in this paper is based on the currently known devious ad contents, when other devious ad contents and distributions scenarios are uncovered, each module can be appropriately extended to take them into account. In the remainder of the section, we describe in detail our approach for implementing each module.

4.1 Network Traffic Collection

The TCM module implements the first step in the *MadDroid* framework. Its objective is to harvest all the network traffic that is involved in operations for delivering ads on a given app. This traffic carries not only data from exchanges between ad networks and the home page view (i.e., when the app is being loaded), but also data from exchanges between the home page and the advertiser's content server (i.e., when the user interacts with the ad). Thus, given an Android apk file, TCM must visit all app UI pages where ads are likely to be loaded, and then explore an interaction with such ads to collect data in the reached landing page.

For scalability reasons, TCM must implement an effective and automated strategy for covering all ad views in an app. Nevertheless, although it is labor-intensive and time-consuming to implement the exploration manually, it is also non-trivial to achieve automation via traditional automated app testing. Indeed, state-of-the-art approaches in Android, such as MonkeyRunner [26], generate random test cases that are not ad-specific: the majority of dynamic execution scenarios will then be wasted for exploring irrelevant UI states. As empirically demonstrated by Suman Nath [58] on a set of ad-supported apps, over 90% of the automatically explored UI states are not ad-involved pages.

To overcome the efficiency challenge in rapidly and quasi-exclusively focusing on relevant UI states, we propose to tune the exploration strategy by generating ad-intensive test cases, i.e., by favoring ad views. We refer to it as an *ad-first exploration* strategy. We build on the finding of a recent study [58] that most ads are displayed in the main UI page and on the exit UI page. Our ad-first exploration strategy thus attempts to prioritize the views of these pages, and further rely on a breadth-first search algorithm where the views in a page are reordered, i.e., ad views are prioritized.

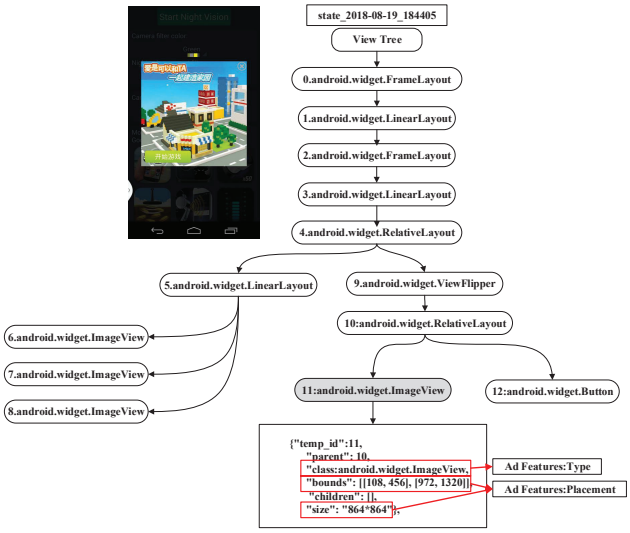

Figure 5: An example of a view tree.

Views are identified by traversing the nodes in a view tree that can be obtained from a given UI state (i.e., a GUI page at a given time in app execution). Figure 5 shows an example of a view tree. The root node represents the base *layout view* on top of which upper views are placed. Parent nodes are containers to child nodes that are subject to users' manipulations. Each node is tagged with basic view information such as position, size, class name, etc. Inspired by a most recent work [31], we use such attributes to identify which nodes among the leaf nodes are likely to be ad view nodes. Specifically, during our exploration, to ensure good coverage, we prioritize and click on each view that falls in the class of WebView, ImageView or ViewFlipper.

Based on the results of the ad-first exploration, dynamic execution of ad-related UI states will lead to a large collection of network traffic. Unfortunately, at this stage, the collected traffic contains not only ad-related traffic but also non-ad related ones (such as data exchanged for app analytics). There is hence a strong need to precisely distinguish between ad and non-ad traffic, in order to correctly extract ad content. To this end, we propose a framework runtime hooking approach to achieve this purpose. Details will be given in the next subsection.

4.2 Ad Content Extraction

The CEM module analyzes the traffic collected through TCM in order to extract relevant content for further assessment. Indeed, by default, TCM collects any traffic that occurs while the ad is being loaded or after it is clicked. Since we are interested in traffic

carrying ad contents, CEM must dismiss all traffic that is not related to advertisements (e.g., parallel traffic from core app functionality). To this end, the first step taken in CEM is to identify ad-related HTTP requests/responses, considering those that are done as part of exchanges with ad networks. Take Table 1 as an example, a simplified list of HTTP requests harvested from the execution of app *com.bbsoft.InternetPolyglot* is illustrated and *startappservice.com* is known as an ad-domain. The first step is hence to highlight such ad-domain related HTTP requests (cf. lines 1, 3 and 4).

Table 1: Simplified list of harvested HTTP requests (domain + path)

1	AD-DOMAIN	info.static.startappservice.com /1.4/getadsmetadata
2	NON-AD	data.flurry.com /aap.do
	//ad-load	
3	AD-DOMAIN	req.startappservice.com /1.4/gethtmlad
4	AD-DOMAIN	imp.startappservice.com /tracking/adImpression
	//ad-click	
5	NON-AD	cl.untildogtop.com /t/clk
6	NON-AD	my1trk.com /redirect/action /1InYjNywuJnNnYTwiKHNmf3BlZ2E_eQ_Pyi
7	NON-AD	www.spyoff.com /geo

As shown in Fig. 6, given an ad-domain whitelist, it would be straightforward to pick out ad-load traffic from a collection of network traffic. Unfortunately, it is not easy to manually build such a whitelist of ad domains, mainly due to two reasons. On one hand, there are a plethora of ad libraries and new ad networks might continuously join the ecosystem. On the other hand, we empirically found that, for a given ad library, the domain names of ad networks may change, and even one ad library may correspond with a number of domain names, making it hard to label a complete and accurate list of ad-domain names. For example, we found that the ad network "daoyoudao" [23] has dozens of ad-domain names, including "daoudao.com", "guiji.com", "133155.com", "161161.com" and "150155.com", etc.

Therefore, we propose to develop in CEM a runtime HTTP hooking approach (as shown in Fig. 6) for iteratively identifying ad relevant domain names, so as to locate ad-relevant traffic. Our approach dynamically hooks all the HTTP-related methods at the framework level. Following the same ad-first exploration approach detailed in the previous section, when a HTTP-related method is reached, the hooking module will record the current execution stack trace and the URL associated with the HTTP method. Following the dumped stack trace, our approach can automatically locate the package that initiates the HTTP connection and build a mapping (hereinafter referred to as pkg-domain mapping) from packages to URL domains. If the package belongs to a known ad library, all the domains triggered by this package will be regarded as ad-domains and recorded into the mapping. Similarly, if the domain matches one of the ad-domains recorded in the mapping, the corresponding package will be flagged as an ad library and hence recorded into the mapping. By doing so, the runtime hooking approach enables our approach to iteratively grow the whitelist of ad-domains.

Once ad traffic is located from the network traffic collected by TCM, it can unfold the next step of extracting ad contents from all

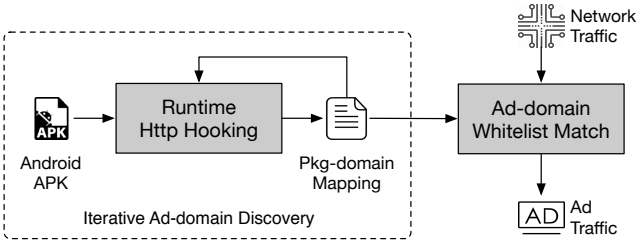

Figure 6: Runtime hooking approach for locating ad traffic.

relevant ad-response messages (i.e., those sharing the same session id as the identified ad request messages). Ad contents that are extracted include images and executable scripts. Such ad traffic that is only relevant to simple message exchange (e.g., sending a message to confirm an ad impression) without carrying actual ad content will be ignored (e.g., lines 1 and 4 in Table 1 will be ignored).

With regards to ad click, the response message of an ad-loading request generally includes a URL indicating the target address when the ad is clicked. Indeed, ad clicking requests are supposed to be redirected to the appropriate content server whose domain address is stored in the ad content. More specifically, the domain address is bound to ads' click events. Let us take Listing 1 again as an example, after the ad is clicked (line 5), an ad clicking request will be sent to a content server, which returns a redirection link (line 6) that eventually leads to the ad landing page (line 7), which is a VPN app website. By analyzing the binding information, we can retrieve this address and subsequently identify ad clicking-related traffic. By dynamically exploring the ad clicking events on installed apps, we can further collect three types of ad contents: (1) *redirection links*: the URL bound to the ad click event might not be the final destination: i.e., the landing page may be reached after several redirections. (2) *downloaded APKs*: the ad clicking request will trigger a downloading process of non-requested apps. (3) *Google Play pages*: the ad click will be directed to Google Play to promote the advertised app.

There are at least two means to explore the ad clicking events: (1) by simulating the clicking request (e.g., record the request URLs and then send requests using a browser later) or (2) by actually clicking the ad. The latter approach is adopted in this work as we have experimentally found that the former approach is likely leading to failures of requests. For example, we have empirically observed that some redirection links are time-sensitive. The emulated request after a certain time period will simply result in an invalid request.

4.3 Ad Deviousness Detection

The DDM module in *MadDroid* is the core component in charge of implementing analysis procedures for assessing the variety of artifacts collected by CEM in order to check against the presence of any devious ad content. Given that each group of devious ad content presents specific characteristics and detection challenges that require specific detection schemes, we design DDM with a plugin-based system architecture. This offers the flexibility to address newly appearing groups of ad contents by integrating an independent plugin implementing the required analysis of ad contents using specialized state-of-the-art techniques.

In the current version of the *MadDroid*, we have already proposed prototype plugins that cover the devious ad content groups. We now detail, for each plugin, the detection strategy that was applied as well as some implementation details.

4.3.1 ***Click-deceptive Image.*** The main idea is to check whether the image actually embeds a "cross" symbol. This refers to the problem of recognizing objects in images. Traditional object detection algorithms have shown to be effective for object recognition [10, 12, 61]. In this work, we adopt the YOLO (You Only Look Once) approach, which is proven to have achieved higher efficiency and accuracy than other approaches [12, 61]. The work-process is illustrated in Fig. 7. First, the algorithm splits the image into a $S * S$ grid, where each cell is called a bounding box. Then, it predicts the probability and confidence of each box to be the object that should be recognized. The output of this step is a tensor of $S \times S \times (B \times 5 + C)$ dimension, where B is the number of bounding boxes capable of marking the object: we set $B = 1$ in our work as we assume that a single bounding box is enough to fully contain the "cross" symbol. C is the number of objects to be recognized: $C = 1$ in this work as we aim to recognize only a single object. Finally, YOLO uses a non-maximal suppression approach to choose the box that yields the best prediction score. For more details on the inner-working of the algorithm, we refer the reader to the description in [12, 61].

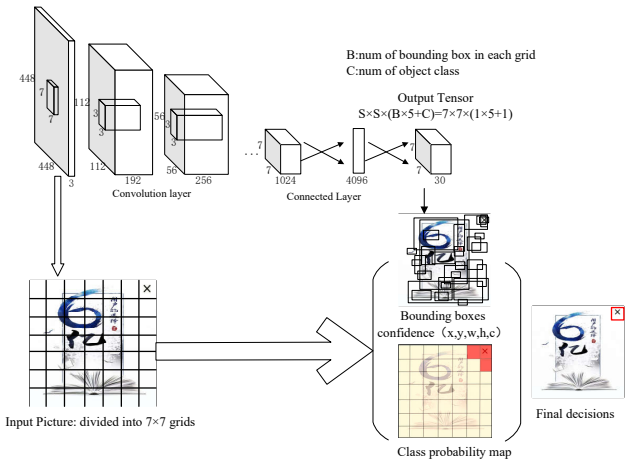

Figure 7: Click-deceiving picture detection based on YOLO.

Although object recognition techniques have been proposed for decades in various applications, including face detection, the literature, to the best of our knowledge, does not report any work related to the case of "cross" (×) symbol, a simple but pervasive object. As a result, there is no public dataset that we can leverage to train our model for the detection of ad click-deceptive Images. As part of the *MadDroid* effort, we propose to construct such a training set from scratch. Although we had already harvested some sample images during our manual investigations for the purpose of characterizing devious ad contents, the obtained set is not representative. Since normal ads also contain "cross" symbol and it is difficult for users to distinguish if the "cross" symbol is displayed from an independent image, we collect images from found ads and artificially embed "cross" symbols into them. Note that we have collected more than 100 different kinds of "cross" symbol images from normal ads, and we artificially embedded them into images with random positions and random size (within a normal size range). Eventually, we obtain a set of 2,375 pictures and record the ground truth (i.e., the actual position of the "cross" symbol) via a common object recognition format `PASCAL Visual Object Classes` [9].

4.3.2 ***Censored Image.*** We treat separately the cases of gambling and pornographic/violence/medical devious ad content which are all considered as censored images.

Pornographic/Violence/Medical Ad Picture. Image detection has been a hot topic in the research community for decades. With the recent advances in CV and deep learning, the research line has matured, and many highly effective approaches [7, 47, 52, 67] are available. Given as an input an ad image, Google Vision API [34] will output a range from 1 to 5 (i.e., from very unlikely to likely and very likely) indicating the likelihood of being the image targeted by the analysis (e.g., pornographic, violence, or medical). In this work, we consider that a given image is a censored one as long as the prediction result is equal or higher than 4, indicating the image is likely or very likely to be a pornographic/violence/medical image.

Gambling Ad Picture. Because of the heterogeneity in gambling (e.g., blackjack, poker, etc.), it is hard to build a graphical model that captures the "gambling" instances accurately. Thus, instead of detecting gambling images graphically, we adopt a simple approach that focuses on the text embedded in the ad images. To that end, we rely on Optical Character Recognition (OCR) [8] techniques to extract any text from a given ad image and match them against a predefined set of gambling keywords. We consider that an ad is about delivering gambling contents if any of its embedded text tokens match any of the keywords enumerated currently in the prototype plugin implementation. The gambling-related keywords are collected from various sources, including the top searched Casino keywords in Google [57], and the frequently presented words on several online gambling websites (in both English and Chinese). Since similar ad contents displayed on online gambling websites might be also used in mobile apps, when counting the recurrently presented words, we also take into account the words presented in pictures of those websites. Eventually, our gambling-related keyword set contains 100 words in Chinese and English[2].

4.3.3 ***Other Devious Ad Content Groups.*** Devious ad content for other groups, namely *Malicious Script*, *Malicious Redirection Link* and *Malicious App*, which, contrary to images, have been well investigated in the security community. Hence, given that we build a framework, for our plugin prototypes, instead of reinventing the wheels, we leverage state-of-the-art techniques to detect issues with such artifacts. Specifically, we rely on anti-virus scanners to flag malicious artifact. Concretely, DDM sends these non-image artifacts to VirusTotal [11], a free online service that integrates over 60 anti-virus engines, has been widely adopted by the research community [17, 48, 65]. Our prototype plugins implement detection schemes where, for each artifact that is sent to VirusTotal will be considered as ad devious content whenever at least three (3) anti-virus scanners flag it as suspicious.

[2]Example keywords include gambling, casino, Macau dealer, beauty Croupier, lottery, GoldenFlower (ZHAJINHUA in Chinese), etc.

4.4 Implementation

The core of the *MadDroid* framework is implemented in Python. It includes the architectural foundation for gluing the input and output formats of the different modules, as well as for reporting decisions. We have implemented a lightweight UI-guided test input generator to dynamically explore Android apps, with a special focus on ad views during exploration and enforced the ad-first exploration strategy. The network traffic is harvested through Fiddler [66], which logs all HTTP(S) traffic and supports customized Fiddlerscripts [41] to extract specific contents from the traffic. The runtime hooking is based on the Xposed framework [1], which can collect the runtime information of tested Android apps.

5 EVALUATION

Since *MadDroid* aims at automatically harvesting mobile ad contents and detecting devious ad contents, we evaluate it by answering the following research questions.

RQ1: Can *MadDroid* detect devious mobile ad contents?
RQ2: How effective is the HTTP hooking approach (in the CEM module) in locating ad traffic from general network traffic?
RQ3: How accurate is *MadDroid* in detecting devious mobile ad contents?

Our experimental setup includes the construction of a large set of ad-supported apps from markets, the execution of these apps on a physical device, and the collection of network traffic.

5.1 Dataset Construction

To prepare the dataset for evaluating *MadDroid*, we resort to the well-known AndroZoo dataset [43] to crawl Android apps. Since we are only interested in apps displaying advertisements, we further leverage VirusTotal to collect ad involved apps, i.e., adware. In this work, we consider a given Android app is adware as long as one anti-virus engine flags it as such. To this end, we randomly collected 40,000 adware from AndroZoo, including 20,000 Google Play and 20,000 third-party apps, to support our experiment.

Among the randomly selected 40,000 apps, we run each app on a Nexus 5 smartphone, and we use six smartphones in parallel. Considering that loading an ad from a remote server may take time, we set the transition time in app automation to 5 seconds. Overall, automated exploration for each app takes on average 2 minutes. It takes roughly ten days to run all the apps automatically. Contrary to prior related work [16, 60], we do not rely on emulators given that ad libraries may implement verification steps to avoid ad networks from serving ads when the app is being experimented on emulated environments [68] (the objective being to prevent fake impressions of ads [22], i.e., unjustified profit for app developers).

5.2 Harvested Ad Content

Ad-related Traffic: Out of the 40,000 apps, we were able to successfully run 38,553 (i.e., 96.38%) on the Nexus 5 smartphones. During the execution of these apps, the TCM module has collected in total 2,488,897 HTTP and HTTPS messages, from which our CEM module flags 541,129 messages related to ad-load (21.7%) and 692,122 messages related to ad-click (27.8%).

Ad Content: The CEM module then extracts ad contents from the collected traffic: we retrieved 83,347 ad images, 52,592 executable scripts, 49,392 redirection URLs, and 2,545 apps directly downloaded and 2,081 apps promoted via Google Play by clicking the ad views.

5.3 RQ1: Overall Results

Devious Content Detection: As detailed in Table 2, the DDM module flags 279 ad images (specifically, 172 adult, 61 medical, 37 gambling, and 9 violence ad images), 112 executable scripts, 1,822 redirection URLs and 1,457 downloaded apps as devious ad contents. These statistics show that ad clicking contents (i.e., obtained by clicking on displayed ads) are more likely to be devious than ad loading contents (i.e., obtained when loading a page with ad view). This is reasonable since dynamic analysis can reveal deviousness if the content is available automatically on the host app (as what ad loading request does) and subsequently may prevent their acceptance on markets. Instead, leaving their loading, at runtime, from third-party servers is a more effective distribution model.

Nevertheless, although devious ad loading contents are more scarce, they may have a higher impact on the security and privacy of end users. Indeed, unlike ad clicking contents, which may not be triggered (e.g., the ad is not clicked), ad loading contents will, in any case, be delivered to users when the app is launched.

Table 2: Statistics on harvested ad contents.

Ad Content	Total	Devious	Type
Ad Images	83,347	279 (0.33%)	Ad Loading
Executable Scripts	52,592	112 (0.21%)	Ad Loading
Ad Redirection URLs	49,392	1,822 (3.69%)	Ad Clicking
Downloaded Apps	2,545	1,457 (57.25%)	Ad Clicking

Malware Distributed by Ad Content: It is noteworthy that more than 57% of the downloaded apps are alerted as suspicious by antivirus engines (hence are categorized as devious content in Table 2). More than 30% of these devious contents are even flagged by over 10 anti-virus engines, indicating a consensus on their maliciousness. Table 3 lists the top 5 identified malware ranked by the number of VirusTotal anti-virus engines that agree on them being malicious. We further resort to Google Play and an ASO website (www.chandashi.com) that contains apps in more than 10 third-party markets to search for these apps (based on their unique identifiers). Expectedly, 21% of them (311/1457) are not hosted on any markets (both official and alternative markets), and over 91% (1332/1457) of them are not listed on Google Play. This result suggests that attackers are leveraging ad contents as a new channel to distribute malicious apps, especially considering that more and more app markets are enforcing strict security checks.

Host Apps of Devious Content: We further look into the host apps of these devious content to investigate the spread of devious ad contents. Results are summarized in Table 4. Roughly 6.02% of apps (2,322 out of 38,553) in our dataset are identified as delivering devious ad contents. The fact that more devious contents are collected than the number of host apps suggests that one app may repeatedly present devious ad contents. Sometimes the same app may present a diversity of devious ad contents. Moreover, even popular apps on the official Google Play market (e.g., the popular "Magic Candy" game app[3]) are involved in providing devious ad

[3]By the time of this study, this app is still available on the Google Play market [13] and has received more than 10 million installs.

Table 3: Top 5 downloaded malicious apps ranked by the number of flagged VT engines.

Package Name	MD5	Source App	# Engines
com.zhulai.jingjimoren	d3a6fa8359ad1b139004e617ce3baab8	com.ziipin.softkeyboard.kazakh	44
girl.game.weaimeng	21846ecdfe3ae93391372bdd1cd43032	air.com.aoaogame.game34	37
afltr.austscf798.zhnf760	e260fd6f711aea632bb4aae2776a1cef	com.easaa.c000000021	31
com.zhulai.xingqiudazhan	ecf78456db04bce595f798c20fb0bf9f	com.ziipin.softkeyboard.kazakh	31
com.dfg57.g8t5g	020a0520df8f5562cd8c2c14ae3ef6ea	com.droidlink.bmw.lock	30

contents (distributing click deceptive images). This is evidence that the identification and blocking of devious ad content remains an unresolved issue in the industry. The research community thus needs to put more effort into approaches and tools for addressing unethical behavior in mobile ads so as to provide a clean and safe environment for displaying mobile ads.

Table 4: Host apps of devious content.

Type	# Devious Contents	# Host Apps
Click-deceptive Image	525	40
Censored Image	279	240
Malicious Script	112	46
Malicious Redirection Link	1,822	838
Malicious App	1,457	1,267
Total	-	2,322

The Role of Ad Networks: We further investigate the distribution of ad networks in terms of the number of devious ad contents that they push to app users' devices. In this work, we have identified in total 3,518 ad host names (or networks in simplicity). Due to space limitation, we only listed the top 3 ad networks that distribute devious ad content for each group, as shown in Table 5. It is interesting to observe that, for censored images, malicious scripts and malicious links, most of them are distributed by popular ad networks. For example, over 46% of malicious links were distributed by startapp and the google ad network. Considering that these popular ad networks are widely adopted, many users may have already been affected by the presence of devious ad contents on their devices. For deceptive images and malicious apps, most of them were found in less-popular ad networks. We argue that the ad networks need to be responsible for such threats by implementing adequate means to keep devious ad contents from being pushed to end users.

The Origin of Devious Contents: We further seek to investigate the advertisers that distribute the devious contents by analyzing the landing page of malicious redirection links and the downloading address of malicious apps. Table 6 lists the top 5 for each of them. We observe that most of the malicious links and malware were originated from several specific domains. Top 5 domains occupied over 23% of malicious links, and over 56% of malware downloading URLs. This result suggested that some advertisers have the tendency to release malicious contents. Therefore, it is important and urgent to identify them and remove them from all the ad networks.

Comparison with the state-of-the-art: A recent closest study [16] characterizes the malicious behavior of mobile ad landing pages. Unfortunately, their tool and dataset are not publicly available. Thus, we can neither apply their approach to the apps we randomly selected in this work nor apply *MadDroid* to their apps. Hence, we explain why our approach can collect much more ad contents than theirs according to the design. First, like all the other previous studies, Chen *et al.* only focuses on ad clicking contents, letting ad loading contents untouched. Second, they only take into account

Table 5: Top ad networks that distribute devious ad content.

ad network	# devious content	% devious content
Top 3 ad networks ranked by the number of distributed click deceptive images		
me2s.co	382	72.8%
go2s.co	120	22.9%
droidhen.com	8	1.5%
Top 3 ad networks ranked by the number of distributed censored images		
startappexchange.com	146	52.3%
googleads.g.doubleclick.net	34	12.2%
adeco.com	16	5.7%
Top 3 ad networks ranked by the number of distributed malicious scripts		
googleads.g.doubleclick.net	74	66.1%
startappexchange.com	18	16.1%
nads.wuaiso.com	3	2.7%
Top 3 ad networks ranked by the number of distributed malicious links		
startappexchange.com	496	30.1%
googleads.g.doubleclick.net	267	16.2%
mobincube.com	155	9.4%
Top 3 ad networks ranked by the number of distributed malware		
ie2o.com	343	23.5%
gamezi.com	217	14.9%
td68x.com	132	9.2%

Table 6: The Origin of Devious Contents.

domain	# malicious link	% malicious link
Top 5 landing page domains of malicious redirection links		
revcontent.com	145	8.0%
take-your-prize-now1.life	119	6.5%
ds-club.ru	75	4.1%
wolve.pro	42	2.3%
inhabitny.com	39	2.1%
Top 5 downloading domains of malicious apps		
domain	**# malicious apps**	**% malicious apps**
ie2o.com	343	23.5%
gamezi.com	217	14.9%
td68x.com	132	9.2%
clouddn.com	78	5.4%
cmbst.cn	54	3.7%

WebView widgets for inferring advertisements. However, we experimentally find that WebView is only used by roughly half of the advertisements (51.93%). *ImageView and ViewFlipper widgets have also been frequently leveraged to display ads.* Third, they exclude all apps with multiple WebViews from their dataset. Our experiment reveals that around 30% (2,751/8,604) of the apps are involved in two or more distinct ad widgets, while 48.8% (1,343/2,715) of them contains two or more Webviews. Finally, the list of ad hosts considered by Chen et al. also limits their capability of identifying all advertisements. As shown in the next section, our HTTP hooking approach can significantly increase the list of ad hosts for identifying advertisements.

5.4 RQ2: Effectiveness of the HTTP hooking

The CEM is a key module where non-ad traffic is filtered out through a HTTP hooking approach. Given that this step is essential to locate and extract ad contents, it is important to assess its effectiveness so as to validate this step in the workflow. Recall that the HTTP hooking approach takes as input a set of ad libraries and/or ad hosts and the library-host mapping is built iteratively. We evaluate our approach through the following three settings.

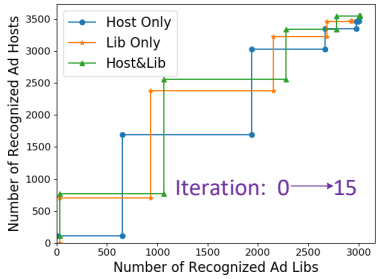

Figure 8: Results of RQ2 in 15 Iterations. Each iteration is denoted by an edge: vertical edge for expanding ad hosts (based on the latest ad library list) while horizontal edge for expanding ad libraries (based on the latest ad host list).

- **S1: Ad libraries only.** Wwe send only ad libraries (with the ad host set as empty) to evaluate the effectiveness of the HTTP hooking approach. Specifically, 52 popular ad networks, maintained by LibRadar [51], are considered.
- **S2: Ad hosts only.** Instead of giving ad libraries as input, we send only ad hosts to run the experiment. Specifically, the 1,315 ad hosts[4] leveraged by Chen *et al.* [16] in their mobile advertising threats study are considered in this experiment.
- **S3: Ad libraries and hosts.** Finally, we take into account both the aforementioned 52 ad libraries and the 1315 ad hosts as input to conduct the evaluation.

Figure 8 illustrates the experimental results. The *x-axis* and *y-axis* represent respectively the number of ad libraries and ad hosts. Interestingly, no matter starting from which setting, all the experiments tend to converge to the same result within 15 iterations. Thanks to the HTTP hooking approach, the number of ad hosts has almost tripled, resulting in around 3,500 ad hosts, which in turn immensely increases the collection of ad contents by 126%.

Figure 9 presents the top five involved ad libraries and ad hosts, w.r.t. the number of hosts triggered by each library and the number of libraries triggering the same host, respectively. The top-ranked library, namely *com.applovin*, is even associated with 357 distinct ad hosts, while the top-ranked ad host, namely *googleads.g.doubleclick.net*, is triggered by 175 ad libraries. It is surprising that one ad library can trigger multiple distinct ad hosts and one ad host can be triggered by multiple ad libraries. Our in-depth manual investigation reveals that those top-ranked libraries have usually embedded with multiple other ad libraries and the actual ad requests are triggered by those embedded ones, resulting in hence different ad hosts mapping to different ad libraries.

Our in-depth analysis further reveals that some libraries pointed to the same ad hosts are actually the same ones but have been deeply obfuscated. For example, *api.airpush.com* is a well-known ad library. In this experiment, we find various libraries such as *com.avtqk.ubjir220086* and *com.filGh.hXwrF124710* that trigger the same ad host (i.e., *api.airpush.com*) and these libraries are actually the obfuscated versions of the original *airpush* library. This result suggests that our HTTP hooking approach can be even a promising approach for identifying obfuscated libraries.

[4]This list is continuously being updated. At the time of the study presented by Chen *et al.* [16], the number is 1,183.

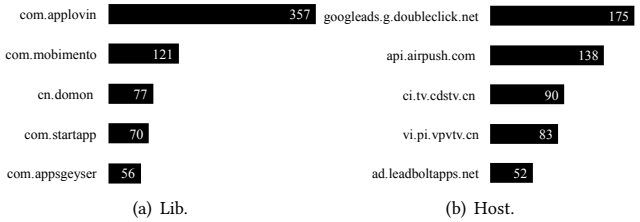

Figure 9: Top five involved ad libraries and ad hosts.

5.5 RQ3: Performance Evaluation

5.5.1 Click-deceptive Image. Given that we have built ourselves the training dataset for detecting click-deceptive ad images, and selected a technique that has not been applied in the literature for such cases, we evaluate the performance of this detection as the main pain point in the validation of the DDM module. We recall that we have collected 2,375 click-deceptive images as introduced in Section 4.3. To evaluate the performance of our plugin for detecting click-deceptive images, we randomly select 2,175 images to form a training set for YOLOv2. Then, we put the remaining 200 click-deceptive images, along with 200 normal ad images (i.e., without "cross" button embedded) into a testing set and apply the trained YOLOv2 model to it. Table 7 summarizes the prediction results of our approach to the detection of click-deceptive images. Among the 400 images in the testing set, our YOLOv2 model flags 201 and 199 images as click-deceptive and normal, respectively. With 5 false positive results and 4 false negative results, our approach yields a precision and recall of 97.51% and 97.99% respectively for predicting click-deceptive images, demonstrating that our approach is quite reliable for recognizing the "cross" button embedded in images.

Most of the closing buttons in ad images follows the form of a cross symbol, but exceptions exist that some of the closing buttons are demonstrated in text images. To this end, we implement a keyword match on the result of OCR that includes close, exit, skip in Chinese and English. Eventually we identify one such case.

5.5.2 Censored Images. Scenarios for detecting Pornographic, Violence, and Medical images are directly based on the popular Google Vision API with pre-trained deep learning models. Google Vision API has been widely used by state-of-the-art approaches and has been experimentally demonstrated to be effective in flagging pornographic, violence, and medical images [16, 18, 30, 55]. As experimentally demonstrated by Chen *et al.* [16], Google Vision API can indeed outperform other image scanning services.

For the gambling image detection, with a set of 100 gambling-related keywords in both Chinese and English (configurable), MadDroid achieves 100% of accuracy for identifying gambling images, as all the identified images indeed contain the defined keywords. Despite that we have formed a relatively large set of gambling-related keywords, it is still possible that some gambling images are overlooked, e.g., they do not contain keywords, or our list of keywords is incomplete. However, these cases are rare in our study.

5.5.3 Malicious Scripts/Links/Malware. Since we are not capable of manually confirming if a given app, redirection link or script is malicious, we rely on VirusTotal to flag malicious ones, which is widely used in our research community.

Table 7: Performance of click-deceptive picture recognition.

	Click-deceptive (Predictive)	Non-Click-deceptive (Predictive)
Click-deceptive (Actual)	True Positive (196)	False Negative (4)
Non-Click-deceptive (Actual)	False Positive (5)	True Negative (195)

6 DISCUSSIONS

Implications. Our findings in this paper suggest that ad networks, even popular ones such as AdMob, are involved in the delivery of devious ad contents to end users' devices. The fact that ad networks are not always delivering legitimate ad contents suggests that the ad contents (likely provided by Advertisers) might not be properly checked by ad networks before being pushed to user devices. As a result, devious Advertisers may exploit the limitations in the current system to advertise devious contents, leading to a poor user experience which harms the reputation of both ad networks and app developers. We argue that ad networks need to introduce automated tools to regulate the behavior of advertisers. Moreover, it is hard to know if ad networks are involved in this black market. They may turn a blind eye on purpose as they have actually hosted the content servers. We hence appeal to the community for putting more effort to explore this new research direction.

Limitations. The implementation of *MadDroid*, however, carries several limitations. First, we take advantage of state-of-the-art app automation technique [45] to explore the app, and use an ad-first exploration strategy, to achieve a balance between time efficiency and ad view coverage, which may cause some ad views to be missed during UI exploration. Nevertheless, our experiments suggest that we could extract much more ad contents than existing studies. Second, our categorization of devious content might be incomplete since it was built based on existing knowledge. Nonetheless, for new types of devious content, it is quite easy to extend the framework of *MadDroid* for further analysis. Third, the ad contents shown in a given app may vary due to factors such as time, location, user identifiers, etc. Thus, in our automation testing, some devious behaviors may not be triggered due to various reasons.

7 RELATED WORK

Mobile Ad Clicking Content Analysis. Beside the state-of-the-art work by Chen *et al.* [16], the closest work related to ours is proposed by Rastogi *et al.* [60], who have experimentally explored the security issues of ad clicking contents, *without considering the ad loading content*. More specifically, they develop a systematic approach to explore the ad contents using a computer vision based app automation technique in an emulator. Comparing to our approach, which explores the ad contents using an ad-first exploration strategy and actually clicking the displayed ads in a smartphone, their approach is more likely to fail, i.e., fewer contents are harvested. Similarly, Son et al. [63] are also interested in the devious behavior of advertisers. They have discovered that malicious advertisers may push executable scripts to access the external storage of user's devices so as to infer sensitive information of users. In this work, our approach has also taken into account the aforementioned three groups of devious ad contents, namely malicious redirection link, malware and malicious script. Moreover, to the best of our knowledge, it is the first work that considered the click-deceptive images and censored images during the loading of mobile ads.

Malicious Web Advertising Analysis. Malicious advertisement has been extensively studied in the context of web advertising, which is so-called web malvertising. This line of studies mainly falls in the group of drive-by-download attack detection. Cova *et al.* [21] and Lu *et al.* [50] proposed to detect drive-by-download attack and malicious Javascripts that embedded in the advertising. s. Stringhini *et al.* [64] and Mekky *et al.* [56] used the properties of HTTP redirections to identify malicious advertisement behaviour. Li *et al.* [46] performed a large-scale study through analyzing ad-related Web traces, and found that malicious advertising infects both top Web sites and leading ad networks (e.g., DoubleClick).

Mobile Ad Fraud Detection. Various research studies are proposed to investigate the fraudulent behaviors of mobile app developers, who aim to entice users to click ads [22, 31, 49]. For example, Crussell *et al.* [22] present an approach called MAdFraud attempting to identify click frauds (fake impressions and clicks). Liu *et al.* [49], by statically analyzing the layouts of apps, have investigated static placement frauds (e.g., too many ads displayed on a single page) on Windows Phone. Dong *et al.* [31], have designed and implemented an approach called FraudDroid to detect fraudulent ads in Android apps. They have empirically summarized nine types of ad frauds, including both static placement frauds and dynamic interactive frauds. Our approach, targeting the devious behavior of advertisers, can be considered as a supplement of these studies towards building a trustworthy and clean ecosystem for mobile advertising.

Mobile Ad Library Detection and Analysis. The majority of research studies targeting the mobile ad ecosystem are actually focused on ad libraries. One line of work focuses on identifying ad libraries [42, 44, 51]. The other line of work focuses on the security and privacy issues of ad libraries [15, 24, 35, 59]. Since ad libraries are normally provided by ad networks, who play an important role in distributing devious ad contents, the aforementioned approaches could be useful for complementing our approach towards better understanding the business model of devious mobile ad contents.

Automated App Testing. In this work, the ad contents are harvested based on automated app testing, where an ad-first exploration strategy is adopted to explore app pages with a specific focus of displayed ads. Automated app testing has been widely adopted for exploring apps at runtime [20, 39]. Several Android UI exploration frameworks [27, 28, 36] have been developed to facilitate app testing. In this work, we build on top of DroidBot [45], and we adopt a sophisticated exploration approach that is originally proposed by Dong *et al.* [31] for generating ad-focused test inputs. Nonetheless, we have a different focus in this work, which is to harvest ad contents from mobile ad networks and advertisers.

8 CONCLUSION

In this paper, we perform a large-scale exploratory study of mobile ad contents, which has been overlooked by the research community. We first create a comprehensive categorization of devious mobile ad contents, then we build *MadDroid*, a framework for automated detection of devious mobile ad contents. By applying *MadDroid* to 40,000 Android apps, we find that devious ad contents are prevalent: 6% of apps in our study are identified as delivering devious ad contents. To the best of our knowledge, *MadDroid* is the first attempt towards mitigating threats from both ad-load and ad-click introduced by mobile ad contents.

REFERENCES

[1] Xposed framework api. https://api.xposed.info/reference/packages.html, 2016.

[2] Google play store malware targets porn ads at millions of kids. http://www.itpro.co.uk/malware/30294/google-play-store-malware-targets-porn-ads-at-millions-of-kids, 2017.

[3] Malware displaying porn ads discovered in game apps on google play. https://research.checkpoint.com/malware-displaying-porn-ads-discovered-in-game-apps-on-google-play/, 2017.

[4] Ad contents contain malicious coinminer scripts. http://bbs.360.cn/thread-15338398-1-1.html, 2018.

[5] Detection result. https://www.virustotal.com/#/file/89225036f339ac1011 80699d85eef790e3017f1d0773d6e4a69e680a2bd27060/detection, 2018.

[6] Developer policy center: Monetization and ads. https://play.google.com/about/monetization-ads/, 2018.

[7] Open nsfw model, 2018.

[8] Optical character recognition - wikipedia. https://en.wikipedia.org/wiki/Optical_character_recognition, 2018.

[9] The pascal visual object classes homepage. host.robots.ox.ac.uk/pascal/VOC/, 2018.

[10] Speeded up robust features. https://en.wikipedia.org/wiki/Speeded_up _robust_features, 2018.

[11] Virustotal. https://www.virustotal.com/, 2018.

[12] Yolo: Real-time object detection. https://pjreddie.com/darknet/yolo/, 2018.

[13] Magic candy - google play. https://play.google.com/store/apps/details?id= com.gamoper.magiccandy.free, 2019.

[14] AppBrain. Current number of android apps on google play. https://www.appbrain.com/stats, 2018.

[15] Michael Backes, Sven Bugiel, and Erik Derr. Reliable third-party library detection in android and its security applications. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 356–367, 2016.

[16] Gong Chen, Wei Meng, and John Copeland. Revisiting mobile advertising threats with madlife. In *The World Wide Web Conference*, pages 207–217. ACM, 2019.

[17] Kai Chen, Xueqiang Wang, Yi Chen, Peng Wang, Yeonjoon Lee, XiaoFeng Wang, Bin Ma, Aohui Wang, Yingjun Zhang, and Wei Zou. Following devil's footprints: Cross-platform analysis of potentially harmful libraries on android and ios. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 357–376. IEEE, 2016.

[18] Shih-Hsin Chen and Yi-Hui Chen. A content-based image retrieval method based on the google cloud vision api and wordnet. In *Asian conference on intelligent information and database systems*, pages 651–662. Springer, 2017.

[19] Geumhwan Cho, Junsung Cho, Youngbae Song, and Hyoungshick Kim. An empirical study of click fraud in mobile advertising networks. In *Proceedings of the 10th International Conference on Availability, Reliability and Security (ARES)*, pages 382–388, 2015.

[20] Shauvik Roy Choudhary, Alessandra Gorla, and Alessandro Orso. Automated test input generation for android: Are we there yet? In *Proceedings of the 2015th International Conference on Automated Software Engineering*, pages 429–440, 2015.

[21] Marco Cova, Christopher Kruegel, and Giovanni Vigna. Detection and analysis of drive-by-download attacks and malicious javascript code. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 281–290, 2010.

[22] Jonathan Crussell, Ryan Stevens, and Hao Chen. Madfraud: Investigating ad fraud in android applications. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pages 123–134. ACM, 2014.

[23] Daoyoudao. Daoyoudao-mobile advertising. http://www.daoyoudao.com/dsp, 2019.

[24] Erik Derr, Sven Bugiel, Sascha Fahl, Yasemin Acar, and Michael Backes. Keep me updated: An empirical study of third-party library updatability on android. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2187–2200, 2017.

[25] Shivang Desai. Malicious android ads leading to drive by downloads. https://www.zscaler.com/blogs/research/malicious-android-ads-leading-drive-downloads, 2017.

[26] Android Developers. Monkeyrunner. https://developer.android.com/studio/test/monkeyrunner/index.html, 2017.

[27] Android Developers. Profile your layout with hierarchy viewer. https://developer.android.com/studio/profile/hierarchy-viewer.html, 2017.

[28] Android Developers. Uiautomator. https://developer.android.com/training/testing/ui-testing/uiautomator-testing.html, 2017.

[29] DIGIDAY. The state of mobile advertising. https://digiday.com/marketing/state-mobile-advertising/, 2017.

[30] Samuel Dodge, Jiu Xu, and Björn Stenger. Parsing floor plan images. In *2017 Fifteenth IAPR International Conference on Machine Vision Applications (MVA)*, pages 358–361. IEEE, 2017.

[31] Feng Dong, Haoyu Wang, Li Li, Yao Guo, Tegawendé F Bissyandé, Tianming Liu, Guoai Xu, and Jacques Klein. Frauddroid: Automated ad fraud detection for android apps. In *The 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2018)*, 2018.

[32] MARIA GERSEN. Mobile ad fraud: Definition, types, detection, 2016.

[33] Google. Healthcare and medicines - advertising policies help. https://support.google.com/adspolicy/answer/176031, 2019.

[34] Google. Vision ai | derive image insights via ml | cloud vision api | google cloud. https://cloud.google.com/vision/#industry-leading-accuracy-for-image-understanding, 2019.

[35] Michael C Grace, Wu Zhou, Xuxian Jiang, and Ahmad-Reza Sadeghi. Unsafe exposure analysis of mobile in-app advertisements. In *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*, pages 101–112. ACM, 2012.

[36] Robotium Developers Group. Robotium. https://github.com/RobotiumTech/robotium, 2017.

[37] Rayna Hollander. Two-thirds of the world's population are now connected by mobile devices. https://www.businessinsider.com/world-population-mobile-devices-2017-9/?r=AU&IR=T, 2017.

[38] MARSHALL HONOROF. Malicious web ad infecting android phones. https://www.tomsguide.com/us/malvertising-lock-android-phones,news-25255.html, 2017.

[39] Pingfan Kong, Li Li, Jun Gao, Kui Liu, Tegawendé F Bissyandé, and Jacques Klein. Automated testing of android apps: A systematic literature review. *IEEE Transactions on Reliability*, 2018.

[40] Selena Larson. Spammy ads that hijack your smartphone are now a virtual plague. https://readwrite.com/2014/05/15/app-redirects-mobile-spam-ads/, 2017.

[41] Eric Lawrence. Understanding fiddlerscript. https://www.telerik.com/blogs/understanding-fiddlerscript, 2013.

[42] Li Li, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. An investigation into the use of common libraries in android apps. In *The 23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER 2016)*, 2016.

[43] Li Li, Jun Gao, Médéric Hurier, Pingfan Kong, Tegawendé F Bissyandé, Alexandre Bartel, Jacques Klein, and Yves Le Traon. Androzoo++: Collecting millions of android apps and their metadata for the research community. *arXiv preprint arXiv:1709.05281*, 2017.

[44] Menghao Li, Wei Wang, Pei Wang, Shuai Wang, Dinghao Wu, Jian Liu, Rui Xue, and Wei Huo. Libd: scalable and precise third-party library detection in android markets. In *Software Engineering (ICSE), 2017 IEEE/ACM 39th International Conference on*, pages 335–346. IEEE, 2017.

[45] Yuanchun Li, Ziyue Yang, Yao Guo, and Xiangqun Chen. Droidbot: a lightweight ui-guided test input generator for android. In *Software Engineering Companion (ICSE-C), 2017 IEEE/ACM 39th International Conference on*, pages 23–26. IEEE, 2017.

[46] Zhou Li, Kehuan Zhang, Yinglian Xie, Fang Yu, and XiaoFeng Wang. Knowing your enemy: Understanding and detecting malicious web advertising. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 674–686, 2012.

[47] Yu-Chun Lin, Hung-Wei Tseng, and Chiou-Shann Fuh. Pornography detection using support vector machine. In *16th IPPR Conference on Computer Vision, Graphics and Image Processing (CVGIP 2003)*, volume 19, pages 123–130, 2003.

[48] Martina Lindorfer, Matthias Neugschwandtner, Lukas Weichselbaum, Yanick Fratantonio, Victor Van Der Veen, and Christian Platzer. Andrubis–1,000,000 apps later: A view on current android malware behaviors. In *Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS), 2014 Third International Workshop on*, pages 3–17. IEEE, 2014.

[49] Bin Liu, Suman Nath, Ramesh Govindan, and Jie Liu. Decaf: Detecting and characterizing ad fraud in mobile apps. In *NSDI*, pages 57–70, 2014.

[50] Long Lu, Vinod Yegneswaran, Phillip Porras, and Wenke Lee. Blade: An attack-agnostic approach for preventing drive-by malware infections. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, pages 440–450, 2010.

[51] Ziang Ma, Haoyu Wang, Yao Guo, and Xiangqun Chen. Libradar: fast and accurate detection of third-party libraries in android apps. In *Proceedings of the 38th international conference on software engineering companion*, pages 653–656. ACM, 2016.

[52] Jorge A Marcial-Basilio, Gualberto Aguilar-Torres, Gabriel Sánchez-Pérez, L Karina Toscano-Medina, and Hector M Perez-Meana. Detection of pornographic digital images. *International journal of computers*, 5(2):298–305, 2011.

[53] Huawei Market. Huawei market app developer policy. http://developer.huawei.com/consumer/cn/devservice/develop/mobile, 2018.

[54] Tencent Myapp Market. Tencent myapp market app developer policy. http://open.qq.com/, 2018.

[55] Masoud Mazloom, Robert Rietveld, Stevan Rudinac, Marcel Worring, and Willemijn Van Dolen. Multimodal popularity prediction of brand-related social media posts. In *Proceedings of the 24th ACM international conference on Multimedia*, pages 197–201. ACM, 2016.

[56] H. Mekky, R. Torres, Z. Zhang, S. Saha, and A. Nucci. Detecting malicious http redirections using trees of user browsing activity. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, pages 1159–1167, April 2014.

[57] Mondovo. The most searched casino keywords in google | mondovo. https://www.mondovo.com/keywords/casino-keywords, 2019.
[58] Suman Nath. Madscope: Characterizing mobile in-app targeted ads. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pages 59–73. ACM, 2015.
[59] Paul Pearce, Adrienne Porter Felt, Gabriel Nunez, and David Wagner. Addroid: Privilege separation for applications and advertisers in android. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, pages 71–72. Acm, 2012.
[60] Vaibhav Rastogi, Rui Shao, Yan Chen, Xiang Pan, Shihong Zou, and Ryan Riley. Are these ads safe: Detecting hidden attacks through the mobile app-web interfaces. In *NDSS*, 2016.
[61] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.
[62] Rui Shao, Vaibhav Rastogi, Yan Chen, Xiang Pan, Guanyu Guo, Shihong Zou, and Ryan Riley. Understanding in-app ads and detecting hidden attacks through the mobile app-web interface. *IEEE Transactions on Mobile Computing*, 17(11):2675–2688, 2018.
[63] Sooel Son, Daehyeok Kim, and Vitaly Shmatikov. What mobile ads know about mobile users. In *NDSS*, 2016.
[64] Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. Shady paths: Leveraging surfing crowds to detect malicious web pages. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 133–144, 2013.
[65] Guillermo Suarez-Tangil and Gianluca Stringhini. Eight years of rider measurement in the android malware ecosystem: Evolution and lessons learned. *arXiv preprint arXiv:1801.08115*, 2018.
[66] Telerik. Fiddler - free web debugging proxy - telerik. https://www.telerik.com/fiddler, 2019.
[67] Adrian Ulges and Armin Stahl. Automatic detection of child pornography using color visual words. In *Multimedia and Expo (ICME), 2011 IEEE International Conference on*, pages 1–6. IEEE, 2011.
[68] Timothy Vidas and Nicolas Christin. Evading android runtime analysis via sandbox detection. In *Proceedings of the 9th ACM symposium on Information, computer and communications security*, pages 447–458. ACM, 2014.
[69] Eliana Vuijsje. Malvertising: A profound threat to mobile advertising. https://www.blog.geoedge.com/single-post/2016/05/10/Malvertising-A-Profound-Threat-to-Mobile-Advertising, 2016.
[70] Wandoujia. Wandoujia (ali app) developer policy. http://aliapp.open.uc.cn/wiki/?p=140, 2018.
[71] Haoyu Wang, Hao Li, Li Li, Yao Guo, and Guoai Xu. Why are android apps removed from google play? a large-scale empirical study. In *The 15th International Conference on Mining Software Repositories (MSR 2018)*, 2018.