DeviceWatch: A Data-Driven Network Analysis Approach to Identifying Compromised Mobile Devices with Graph-Inference

EUIJIN CHOO, University of Alberta, Canada
MOHAMED NABEEL, MASHAEL ALSABAH, ISSA KHALIL, and TING YU, Qatar Computing Research Institute, Qatar
WEI WANG, Beijing Jiaotong University, China

We propose to identify compromised mobile devices from a network administrator's point of view. Intuitively, inadvertent users (and thus their devices) who download apps through untrustworthy markets are often lured to install malicious apps through in-app advertisements or phishing. We thus hypothesize that devices sharing similar apps would have a similar likelihood of being compromised, resulting in an association between a compromised device and its apps. We propose to leverage such associations to identify unknown compromised devices using the guilt-by-association principle. Admittedly, such associations could be relatively weak as it is hard, if not impossible, for an app to automatically download and install other apps without explicit user initiation. We describe how we can magnify such associations by carefully choosing parameters when applying graph-based inferences. We empirically evaluate the effectiveness of our approach on real datasets provided by a major mobile service provider. Specifically, we show that our approach achieves nearly 98% **AUC (area under the ROC curve)** and further detects as many as 6 ∼ 7 times of new compromised devices not covered by the ground truth by expanding the limited knowledge on known devices. We show that the newly detected devices indeed present undesirable behavior in terms of leaking private information and accessing risky IPs and domains. We further conduct in-depth analysis of the effectiveness of graph inferences to understand the unique structure of the associations between mobile devices and their apps, and its impact on graph inferences, based on which we propose how to choose key parameters.

CCS Concepts: • **Security and privacy → Mobile and wireless security;**

Additional Key Words and Phrases: Compromised device, mobile traffic analysis, graph inference

ACM Reference format:
Euijin Choo, Mohamed Nabeel, Mashael Alsabah, Issa Khalil, Ting Yu, and Wei Wang. 2022. DeviceWatch: A Data-Driven Network Analysis Approach to Identifying Compromised Mobile Devices with Graph-Inference. *ACM Trans. Priv. Sec.* 26, 1, Article 9 (November 2022), 32 pages.
https://doi.org/10.1145/3558767

Authors' addresses: E. Choo, 2-21 Athabasca Hall Dept of Computing Science, University of Alberta, Edmonton, Alberta, Canada, T6G 2E8; email: euijin@ualberta.ca; M. Nabeel, M. AlSabah, I. Khalil, and T. Yu, Qatar Computing Research Institute, Qatar, HBKU Research Complex, Doha, Qatar POBOX 5825; emails: {mnabeel, msalsabah, ikhalil, tyu}@hbku.edu.qa; W. Wang, Beijing Jiaotong University, Information Security 3 Shangyuancun, Haidian, Beijing, China, 100044; email: wangwei1@bjtu.edu.cn.

2471-2566/2022/11-ART9 $15.00
https://doi.org/10.1145/3558767

1 INTRODUCTION

ISPs play an increasingly important role in preventative cyber security [9, 21]. One emerging attack vector that can be effectively tackled at the ISP level is the detection of compromised mobile devices. Recently, researchers proposed to identify compromised devices from an ISP's point of view [33, 60, 76]. ISPs have direct access to key network traces and information, which enables them to perform early detection of compromised mobile devices. Once detected, ISPs can inform their customers including organizations so that they can take proper actions [16].

Indeed, organizations have encouraged the use of personal mobile devices in workplaces, increasing the security incidents involving mobile devices. A recent study shows that one in three organizations has faced a security incident due to compromised devices having malicious apps [63]. Such devices may leak sensitive information or conduct unauthorized financial transactions [20, 49, 50, 60, 69]. A key challenge in mitigating such security threats is to accurately detect compromised devices and take action. As organizations have little control over mobile devices and do not have access to all mobile network traffic, it would be more effective to do the detection at the ISP level.

A variety of techniques have been proposed in the literature to detect malicious apps, most of which rely on code analysis [20, 23, 33, 74] or network-based analysis [15, 41, 44, 52, 67]. However, these techniques require the inspection of a vast number of apps created constantly and identifying local features for each device and/or app. Therefore, a different method, that is not only robust but also scales to a large network, is required to detect compromised devices.

Many apps have in-app advertisements promoting other apps or in-app purchases [57]. While using such apps, users are often led to download related apps and potentially fall victim to *drive-by download* attacks [29, 36, 37, 51]. Furthermore, many users also tend to install apps not published in official app stores such as Google Play. For example, in some countries such as China, users are blocked from accessing Google Play and thus have to use other stores with weaker security practices [31, 66]. Users in such alternative marketplaces are often more exposed to potentially malicious apps with targeted recommendations based on their past installation history.

Motivated by the above observation, we hypothesize that there exists a *homophily relationship* between devices and their installed apps so that devices sharing similar apps would have a similar likelihood to be compromised [37, 51]. We thus formulate the compromised device detection problem as a graph-inference based classification task.

For an ISP to apply graph inference to mobile network data, there are four key challenges that need to be addressed including (1) identifying mobile devices and apps from the traffic, (2) deriving meaningful associations that could be applied to detect compromised devices, (3) collecting a small set of groundtruth, and (4) choosing effective graph-inferences algorithms with proper parameters.

To identify mobile devices and apps, we propose a data-driven approach. It is often not straightforward to identify which devices use which apps because the ISP does not have any control over the apps running on devices and multiple apps are often simultaneously active on each device [61]. Motivated by *app fingerprinting* approaches [39, 61], we propose to use source IP (device) and app string/TLS certificates (app) in the traffic (Entity Extraction in Figure 1).

Most activities on mobile devices incur through apps and most apps require network connections between devices and their hosting servers while being downloaded, installed, or executed. We thus propose to leverage device-app associations and model the communication involving devices and apps as a bipartite graph, called a device-app graph, where one side is the set of devices (i.e., users) and the other is the set of apps (Graph Building in Figure 1).

Graph-inference approaches require a small set of groundtruth to train a classifier and expand the limited knowledge to detect unknown compromised devices (i.e., devices having unknown yet

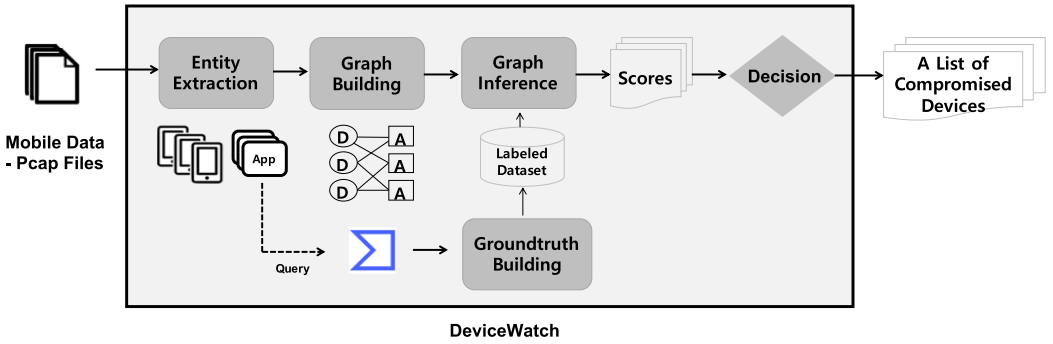


Fig. 1. DeviceWatch workflow.

potentially malicious apps). Given the vast number of unknown apps and little, if any, knowledge on devices in practice, it is often hard to build a ground truth set. We first collect ground truth sets for apps and construct ground truth sets for devices (Groundtruth Building in Figure 1). Then, to *infer* whether an unknown device is compromised, we adapt three well-known algorithms: **Belief Propagation (BP)** [35], **Label Propagation (LP)** [54, 75], and Node2Vec [24] (Graph Inference in Figure 1).

The effectiveness of graph inference depends on the strength of association between nodes in the graph [30]. Unlike other applications where it is relatively straightforward to derive associations (e.g., a malware-infected machine and its activities controlled by command & control servers), it is quite challenging to derive such strong associations between devices and apps due to the fact that: (1) it is often hard for apps to interfere and taint other apps; and (2) user interactions are needed to take any action. Using a 5-terabyte anonymized real mobile network dataset provided by a mobile service provider, We empirically verify our hypothesis that, to a certain extent, there in fact exist associations that can be used to correctly identify compromised devices.

We further study the effect of the relatively weak associations in a device-app graph on graph inference. In doing so, we focus on the behavior of BP, as it performs best on our device-app graph. Concretely, we provide in-depth analysis on the topological similarity and differences between a device-app graph and domain-IP resolution graphs from active DNS data [30] and their impact on the behavior of BP. Given the unique graph structure of a device-app graph, we suggest how to choose proper BP parameters to magnify the relatively weak associations.

Finally, we investigate if the detected devices exhibit undesirable behavior in terms of privacy leakage and access to risky domains and IPs.

In sum, we make the following main contributions. First, we propose *DeviceWatch*, a data-driven approach with graph inference for ISPs to identify **unknown compromised devices at large scale**. In doing so, we propose to leverage an intrinsic association between mobile devices and their apps. To the best of our knowledge, this is the first attempt to use such associations to detect compromised mobile devices. The key advantage is that it is independent of app/device-specific features such as devices' models, versions, or app types (e.g., phishing, malware) and it only requires a small set of groundtruth to operate. *DeviceWatch* thus detects unknown compromised devices at a large scale without time-consuming investigation on individual devices.

Second, we propose a fingerprinting approach that uses source IPs and app string/TLS certificates to identify devices and apps from network traffics, the results of which enable us to build a device-app graph.

Third, through experiments over a large-scale real-world dataset, we show that our approach can effectively detect compromised devices in multiple aspects: ROC-AUC, Precision-Recall,

coverage (Section 4), privacy analysis (Section 5.1), and network infrastructure analysis (Section 5.2). Specifically, we achieve nearly 98% AUC in detecting compromised devices, while we expand our limited knowledge on known compromised devices to further identify as many as 6 ∼ 7 times devices not covered by the ground truth. We show that the newly detected devices, most of whose apps are not known malicious apps, are leaking highly sensitive information in their traffic, and they tend to frequently access risky IPs and domains [4, 25].

Finally, we investigate the unique graph structures of the association between devices and their installed apps, and suggest the choice of key parameters of belief propagation to effectively detect unknown compromised devices.

The remainder of the paper is organized as follows. In Section 2, we review related work. Section 3 describes our proposed approach. Section 4 presents our experimental results. We provide post-analysis on newly detected devices in Section 5. Additionally, we provide a discussion on possible limitations of our approach in Section 6. Finally, Section 7 concludes the paper.

2 RELATED WORK

Malicious App Detection. Malicious app detection research falls into two categories: static/dynamic code analysis and network analysis. Static analysis derives signatures from app binaries based on features drawn from known malicious apps [5, 71]. Dynamic analysis meanwhile monitors app behavior such as privacy leakage or API calls [20, 70]. Unlike PCs, however, it is often hard to perform run-time analysis on mobile devices due to limited resources [38]. Network-analysis approaches detect anomalous network footprints using traffic patterns such as packet size [15, 52, 67, 73]. However, since these approaches often depend on each device- or app-specific features that can be easily obfuscated so that they cannot be applied for general purposes.

By contrast, our approach identifies unknown compromised devices through graph inference without relying on device- or app-specific features. It enables a network admin to quickly manage possible threats encountered at a large-scale without doing expensive analyses on each individual device on the network.

Mobile Security. A few research efforts approach mobile security from a network admin's point of view. Lever et al. conducted a large-scale network level analysis of mobile malware using the DNS traffic [33], which shows the infection rate in real traces. However, they do not provide a solution to identify mobile threats. Sharif et al. proposed a proactive approach that predicts if a user is connecting to a malicious domain or web content by observing her mobile browsing behavior [53]. In this paper, we focus on apps causing devices to be compromised rather than domains.

App Fingerprinting. The goal of app fingerprinting is to recognize apps based on the network traffic they generate [17, 39, 47, 61, 62]. Supervised approaches are typically employed with features such as app-identifying tokens in HTTP headers [17, 39] or TCP traffic patterns [57]. Recently, Van Ede et al. [61] proposed a semi-supervised approach by clustering app traffic based on destination features including TLS certificates exchanged between apps and servers. However, this approach is not directly applicable to our problem settings, where most devices run multiple apps simultaneously (violating one of the key assumptions in [61]). We thus propose to employ a data-driven approach with which we identify an app based on app strings and certificates.

Graph-inference Approaches. Graph-inference approaches have been employed in various security contexts, each of which builds a certain type of graph to capture the association between different entities, e.g., file-machine or file-relation graphs [12, 54, 55], host-domain graphs [35, 42], knowledge graphs [40], and domain-IP graphs [30]. In our context, as mentioned before, it is

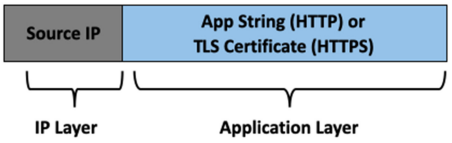


Fig. 2. Simplified IP packet with fields used for data extraction.

often hard to derive strong associations between mobile devices and apps. However, we show that the weak device-app associations can be amplified to successfully identify compromised mobile devices by carefully tuning the underlying graph-inference process. Further, the previous graph-inference approaches often employ one specific algorithm [12, 35, 54] or stated specific parameter values work well without further description [12, 55]. In contrast, we compare different graph inference algorithms, study the unique topological characteristics of device-app graphs, and analyze how such uniqueness may affect the behavior of graph-inference approaches.

The most relevant research to ours is CreepRank which utilizes device-app association to identify creepwares [51]. However, CreepRank only focuses on detecting a special type of apps (i.e., creepwares) on Android devices. Also, CreepRank only considers devices with a security agent (i.e., security-aware users). In contrast, we focus on identifying inadvertent users and their compromised devices without limiting the types of apps or device platforms. Also, our dataset is more comprehensive and representative of typical mobile users, as it is directly collected from an ISP.

3 DEVICEWATCH: THE PROPOSED APPROACH

This work aims to identify unknown compromised mobile devices given a small set of known ones and the network traffic data of mobile devices collected by a service provider. One apparent approach to detect compromised devices is to see whether they have known malicious apps. However, this approach fails to detect compromised devices having previously unknown malicious apps [53]. We thus propose to employ a graph inference approach that can determine whether an unknown device is compromised by analyzing its connections to known devices.

Recall that there are four main challenges we want to address. In this section, we describe how our approach, DeviceWatch, tackles each challenge. Figure 1 illustrates the workflow of DeviceWatch. DeviceWatch (1) first extracts information about devices and apps from traffic (Section 3.1), (2) defines a device-app association (Section 3.2), (3) builds a groundtruth for devices (Section 3.3), and (4) applies graph-inference algorithms to output a final score and to decide the level of compromise for devices (Section 3.4). In the following subsections, we describe each stage of DeviceWatch in more detail.

3.1 Entity Extraction from Mobile Network Traffic Dataset

Our mobile network dataset contains 5-terabytes of 5-days mobile network traffic from a Chinese mobile service provider. Note that ISP does not have any control over the apps running on devices so that it is not straightforward to identify which devices use which apps and build a device-app graph directly from the traffic. In the following, we describe how we extract information about devices and apps from the dataset. Figure 2 shows the fields we use from IP packets. The packet is timestamped. The *app string* and certificate are important in extracting app information; the source IPs and timestamps are important to extract device information.

3.1.1 Device Extraction. Identifying a device from mobile traffic is a challenging problem [33]. Following previous research [33, 39, 57, 61, 69], we consider each source IP address as a device. Note

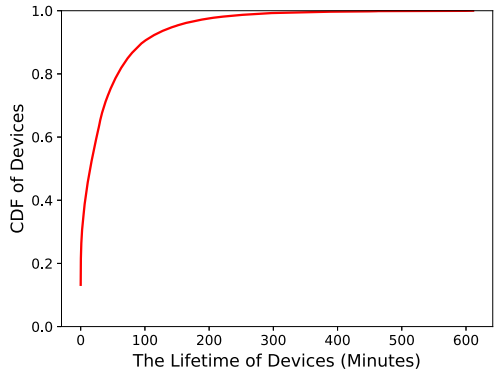


Fig. 3. The lifetime of extracted devices.

Table 1. Example Keywords to Extract Apps

Type	Example Keywords
App String in HTTP header	packagename, appname, apppkg, app_identifier, x-requested-with, apkname, appPkg, appId, user-agent
Domain name in certificates	commonName, organizationName, organizationalUnitName, subjectAltName.

that multiple devices may use the same IP address. Especially, one device may not use the same IP address across multiple days due to the dynamics of mobile IPs [33]. To avoid multiple devices being identified as one device, we further consider a packet daily timestamp following a previous study [33]. Specifically, we define a device using a packet daily timestamp along with the source IP (e.g., 10.x.x.x_2021_08_01). By doing so, we extract 301,428 devices from our traffic. Figure 3 shows the CDF of the lifetime of devices where the x-axis represents the lifetime of devices in the traffic and the y-axis represents the corresponding CDF (i.e., the portion of devices). We measure the lifetime of a device by the difference between the first packet timestamp and the last packet timestamp of the device where there is no gap between packets [57]. The figure shows that 90% of devices have traffic less than 100 minutes and the majority of the lifetime is only a few hours. Given the continuous traffic and a short period of the lifetime, we believe that the identified device is not likely to represent multiple devices.

3.1.2 App Extraction. One may utilize various approaches to identify apps generating the observed traffic [2, 39, 57, 61, 69]. Despite an increasing trend in the usage of HTTPS, developers are allowed to change the configuration to use HTTP [7, 8, 10, 19, 43, 45]. Indeed, our dataset includes both HTTP and HTTPS traffic. We thus utilized both types of traffic to identify more apps and enrich our ground truth. In the following, we describe how we extract app information.

(1) App Extraction using App Strings in HTTP Traffic: This approach extracts app information revealed in HTTP headers. Specifically, we extract the IP packets containing the app string field in the header. We heuristically extract fields containing app strings using keywords. Examples of such keywords are summarized in Table 1. Note that we only present a few examples of keywords, as the keywords are similar yet have small variations such as apppkg and apppkg1.

The app string often contains the name of the app package file, for example:

```
GET /open/confirm.htm?pkg= com.sina.news
```

We assume each unique app string corresponds to an app. By doing so, we gather 5,870 app strings from our dataset.

(2) App Extraction using Certificates in HTTPS Traffic: Recent research suggests that apps can be identified using serial numbers in TLS certificates in HTTPS traffic [61]. From our dataset, we thus extract server certificates to identify corresponding apps. The serial number is unique for each certificate issued by a Certificate Authority. Hence, we consider each serial number as an app. Note that a certificate is indicative of an app only if it is a non-cruise certificate, that is, the certificate contains domains belonging to a single entity. A cruise certificate, usually issued by content delivery networks such as CloudFlare and Akamai, packages many unrelated domains into a single certificate [11]. We inspect the subject fields in a certificate related to a registered domain. Examples of such fields in certificates are summarized in Table 1. Then, we heuristically filter out cruise certificates and identify those non-cruise certificates that are uniquely associated with a single domain. By doing so, we gather 4,501 certificate serial numbers.

To create a unique app name, app developers often use the reverse domain name of the host domain (e.g., *com.youdao.dict* is an app hosted in dict.youdao.com) [18]. Accordingly, for each certificate serial number identified above, we further inspect if the reversed domain name associated with the certificate matches any of the 5,870 app strings. We also query it to 16 popular Chinese app stores [66] to see if it matches any app string in the app stores. If it matches, we replace the certificate serial number with the app string; otherwise, we treat the certificate serial number as an app string. In doing so, we replace 418 certificate serial numbers with app strings, which leaves 4,083 certificate serial numbers used as app strings in HTTPS traffic.

Finally, from the above two steps, we are able to identify 9,953 (5,870 app strings + 4,083 certificate serial numbers) apps.

Each app often has a specific traffic pattern to be used for identification [57, 61]. To investigate the reliability of our app extraction method, we measure if each identified app consistently has similar traffic patterns across multiple devices. Our intuition is that if different apps on different devices are identified as one app (e.g., devices use different apps but all of them have one app string), there should be different traffic patterns across different devices.

Recently, Van Ede et al. proposed to create a signature for an app using destination features such as destination IPs, ports, or TLS certificates, and then they computed the Jaccard similarity between the signature and unknown traffic to classify the traffic [61]. Using the threshold between 0.5 ∼ 0.9, they showed that they could classify an app with acceptable accuracy (0.854 ∼ 0.919). Inspired by [61], we use Jaccard similarity to measure how similar the traffic patterns are across devices given the app string [28]. Note that we focused on HTTP/HTTPS traffic, and thus the destination ports are either 443 or 80 in our dataset. Moreover, by our definition above, each app extracted from HTTPS traffic is associated with one TLS certificate. This implies that the similarity measured based on TLS certificates is 1 across multiple devices given an app. We thus use destination IPs to measure the similarity.

While an app may communicate with various sets of destination IPs depending on the usage, it has been shown that each app often has a unique set of destination IPs used only for the app. For those apps used by more than one device (70% of the total), we thus compare the set of destination IPs to compute a pairwise similarity of traffic patterns among devices having the same app. In doing so, we first filtered destination IPs that are used for multiple apps. Then, we measure the similarity between two devices for an app by the number of the common destination IPs over the total number of destination IPs used for the app. Finally, we define an average traffic similarity of an app by the average pairwise similarities for the app. For example, let us assume we observe the traffic in Figure 4, where three devices have an app called com.app.string. Given the traffic,

Devices	Destination IP	App String
device1	x.x.x.x	com.app.string
	x.x.y.y	
	a.b.c.d	
	e.f.g.h	
device2	x.x.x.x	
	x.x.y.y	
	e.f.g.h	
device3	x.x.x.x	
	x.x.y.y	
	a.b.c.d	
	e.f.g.h	

Fig. 4. Example traffic originated from three devices having the same app string.

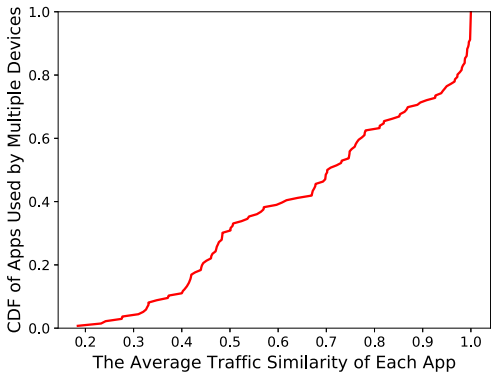


Fig. 5. The CDF of the average traffic similarity.

the similarity values between device1 and device2 are 3/4 = 0.75; between device2 and device3 are 3/4 = 0.75; between device1 and device3 are 4/4 = 1. Then, the average traffic similarity of the app, com.app.string, is (0.75 + 0.75 + 1)/3 = 0.833. Figure 5 shows that the CDF of the average traffic similarity where the x-axis represents the average traffic similarity of each app and the y-axis represents the corresponding CDF (i.e., the portion of apps among those used by multiple devices). We observe that 70% of apps among those used by multiple devices have similarities above 0.5. Note that the app with a low similarity such as 0.3 does not necessarily mean that the app is not correctly identified (i.e., multiple apps are identified as one app), because a certain app may have different traffic patterns depending on devices' models [2, 61]. By default, we thus do not filter any apps based on this similarity measure. On the other hand, a network administrator using DeviceWatch may filter out apps with similarities lower than a certain threshold τ_{sim} depending on her interest (e.g., false positive or false negatives). In Section 4, we will show the performance while varying the threshold.

3.1.3 Assumption and Limitation on Entity Extraction. One of our data limitations is that we do not have physical access to devices and their apps to investigate. Furthermore, multiple devices run multiple apps simultaneously in our traffic. It gives additional challenges to identify a device

or an app from the traffic, compared to related work where research was done in controlled environments [57, 58, 61]. While our dataset reflects the practical scenarios, the limitation is that we cannot directly verify the correctness of our device and app identification. However, note that our work focuses on identifying compromised devices at an ISP level, not pinpointing the exact app of each device. The important assumption we made is that each identified device/app belongs to one entity. While we are confident that the identified device and app indeed belong to one device and one app, respectively, from analyses in Figures 3 and 5, we also assume that a network administrator can further verify using orthogonal approaches, once compromised devices are detected. For example, the administrator may do post-investigation for the detected compromised device if it represents one or multiple devices by checking the phone number associated with the source IP address.

On the other hand, we may not be able to identify certain apps (i.e., some apps on a device will be ignored) if the app string does not present in HTTP, a certificate does not belong to a single entity (e.g., cruise certificate). However, we argue that missing apps on the devices would not affect our approach as our focus is on devices. False negative cases from missing apps will arise only when a device is totally ignored due to no identified apps for the device, or a device is isolated from the graph since its identified apps are not used by other devices. Meanwhile, any prior knowledge about the apps in the network (e.g., an administrator can control running apps in an enterprise network) may increase the coverage of apps and devices, but we consider it as beyond the scope of our study due to our data limitation.

3.2 Bipartite Graph Building

A key challenge to detect compromised devices using an inference algorithm is to first identify meaningful associations capturing graphically homophily relationships. We seek to define an association that is able to create two distinct clusters: compromised and not-compromised devices in the graph.

Mobile users mostly access contents through apps in their devices [58]. Our key insight is thus that there exists an association between a device and its apps following homophily that can be used to identify other compromised devices. Intuitively, if a device installs malicious apps, it is likely to download/install other malicious apps [51]. Meanwhile, most apps require a network connection between devices and their hosting servers while being downloaded, installed, or executed. Thus, we argue that the likelihood of a device being compromised can be measured by analyzing its app usage revealed in the network traffic.

We present a model to reflect the homophily relationship between a device and apps. Specifically, we capture the association between devices and their apps through analysis of network traffic (as an ISP has no direct access to devices) and model such associations as a bipartite graph.

We represent the associations between devices and apps as a bipartite graph $G = (V, E)$ where a set of devices $D = \{d_1, \ldots d_n\} \subset V$ and a set of apps $A = \{a_1, \ldots a_n\} \subset V$ are connected with undirected edges $e(d_i, a_j)$, where d_i is a device and a_j is an app. Figure 6 illustrates an example bipartite graph. Each node in D may belong to one of three categories: not-compromised, compromised, and unknown; and each node in A may belong to one of four categories: benign, malicious, suspicious, and unknown. As illustrated, a compromised device may have edges with all types of apps. An unknown device may have edges with suspicious, unknown, or benign apps. A not-compromised device may have edges with benign or unknown apps, but not with suspicious or malicious apps.

Our bipartite graph drawn from the extracted entities (Section 3.1) includes 1,074,585 edges, i.e., a mapping between devices and apps (either app strings or certificate serial numbers). Figure 7 shows the **Cumulative Distribution Function (CDF)** of the number of devices where the x-axis

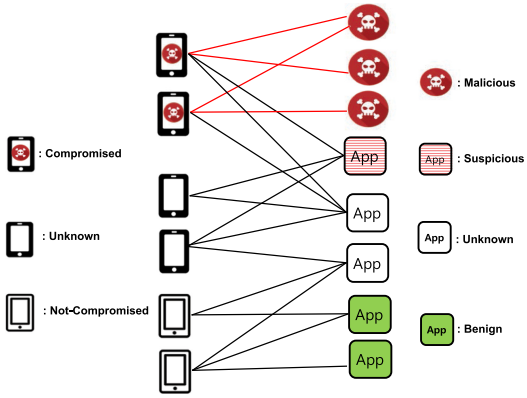


Fig. 6. An example device-app bipartite graph.

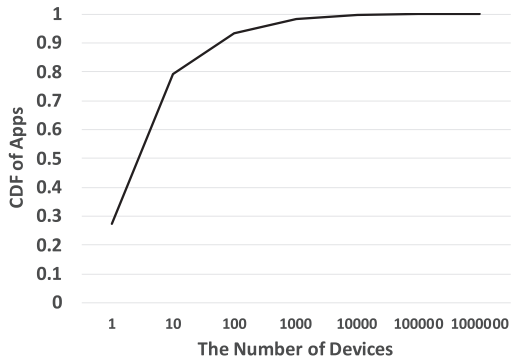


Fig. 7. The CDF of the number of devices for all apps.

represents the number of devices having each app and the y-axis represents the corresponding CDF (i.e., the portion of apps). Note that nearly 30% of apps have only one device. This is mainly because app strings may also include the version, brand, or market names of the apps such as com.sina.news-7.19.3, com.flypig.FindItPro.iphone, and com.bd.superfish.baidu, which we consider individual apps.

Note that we aim to identify unknown compromised devices that may have installed unknown malicious apps. The apps used by most devices, if not all, are not useful for our context because we cannot claim a strong association between a specific device and such apps. For example, even though popular apps such as Facebook are on every device, we cannot claim that all devices having Facebook are equally compromised. Apparently, such apps will lead to false associations, and thus false positives in a graph-based approach. On the other hand, we are more interested in the case where device1 and device2 share specific sets of malicious apps that are not used in many other devices, resulting in a strong association between the set of apps and the possibility of devices being compromised. Similar to previous research using graph inference algorithms [6, 30, 35, 54], we thus exclude popular apps to avoid a number of false positives which can be induced by false association. To do so, we employ a data-driven approach. Concretely, we consider an app popular if it is used by more than N_p number of devices in the traffic. We vary N_p between 1,000 and 10,000, resulting in 1.4% $\sim$ 0.02% of apps being filtered as shown in Figure 7. We shall further discuss the effect caused by this filtering in Section 4.3.1.

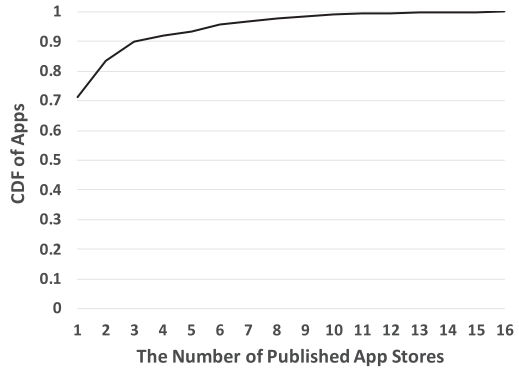


Fig. 8. The CDF of the number of published app stores.

3.3 Ground Truth Sets and Definitions for Detecting Compromised Devices

Our approach requires small sets of ground truth about compromised and not-compromised devices to apply graph inference algorithms. However, given the vast number of unknown apps and little, if any, knowledge on devices in practice, it is often hard to build a ground truth set. We first collect ground truth sets for apps (Section 3.3.1) and construct ground truth sets for devices (Section 3.3.2).

3.3.1 Ground Truth Sets for Apps. Among various intelligence sources, we use VirusTotal (VT) [64] to collect a ground truth set for apps, widely used in the literature [13, 31, 46, 53, 66]. For each query, VT aggregates the responses from more than 50 engines, each of which categorizes the query to *malicious* or *benign*.

To build a ground truth set, we first attempt to download android binaries from 16 popular Chinese Android app stores [66] and iOS app store by searching the app strings extracted in Section 3.1.2. Note that app strings are sometimes not searchable [39, 48] and we cannot pinpoint the exact binaries for 4,083 certificate serial numbers (i.e., not matching to any app strings). We also could not find some app strings matching to iOS app binaries, although certain app strings specifically indicate the iOS app (e.g., com.flypig.FindItPro.iphone). Among 5,870 app strings, 2,367 were found in at least one of 16 app stores. Figure 8 presents the CDF (the y-axis) of the number of app stores (the x-axis). Note that we verify that the binaries are indeed generating the corresponding app strings by capturing network traffic while installing and executing binaries on two mobile devices.[1] We also verify that the reversed app name appeared in certificates in traffic generated by binaries.

We upload the binaries to VT and check whether it is marked as malicious. Note that 29% of app strings are published in multiple app stores, as shown in Figure 8. However, we observe that the maliciousness of each app is the same regardless of the app stores where it is downloaded. This observation agrees with that from prior research: the app string often correctly represents a specific app [59, 66]. Furthermore, recent studies observed that mobile malware does not spread by repackaging as much as before in a large-scale study on Android apps [66, 68]. We thus argue that it is reasonable to rely on the app string to identify and evaluate each app.

Among 2,367 binaries, 1,711 apps are flagged as malicious by at least one VT engine and 656 apps are not flagged by any engine. Previous research suggests that evaluation based on VT may have limited coverage or noise due to multiple reasons [13, 27, 32, 66]. To reduce potential false

[1]Samsung Galaxy Note4 and Sony Xperia Z3 Dual.

Table 2. The Number of Bad Apps with Varying vt and N_p Thresholds

App popularity \ VT	$vt = 3$	$vt = 4$	$vt = 5$	$vt = 6$	$vt = 7$
Not-filtered	1195	1060	955	845	764
$N_p = 10000$	1190	1056	951	841	761
$N_p = 5000$	1189	1056	951	841	761
$N_p = 1000$	1178	1047	943	833	753

Table 3. The Number of Devices with Varying vt and N_p Thresholds

App popularity \ VT	$vt = 3$	$vt = 4$	$vt = 5$	$vt = 6$	$vt = 7$	good device
Not-filtered	45075	27522	24878	22853	15810	143552
$N_p = 10000$	9052	6595	5619	4822	3348	92099
$N_p = 5000$	7644	5962	5004	4245	3348	78820
$N_p = 1000$	3050	2526	2153	1586	968	48999

positives, we label each app using thresholds as follows: If an app is detected as malicious by more than or equal to vt number of engines among the 60 VT engines, we label the app as **bad**; if an app is detected as malicious by fewer than vt engines, we label the app as **suspicious**; if an app is not detected as malicious by any engine, we assume that the app is **good**; if we are not able to find corresponding binaries, we consider the app as **no-info**.[2]

Table 2 summarizes the number of bad apps with various vt and N_p thresholds.

3.3.2 Ground Truth Sets for Devices. Given the ground truth set for apps, we define a **bad device** as one using more than or equal to $N(A_b)$ number of bad apps, where A_b is the set of bad apps; we define a **good device** as one not using any bad or suspicious apps. With consideration of the noise of VT, we take a conservative approach to use $N(A_b) = 2$ as default. Table 3 summarizes the number of devices given the ground truth set for apps.

3.4 The Graph-Inference Based Approach

To determine whether a device is compromised, we follow the guilt-by-association principle [30, 35, 51, 53, 55] which estimates the *guiltiness* of a node by propagating the prior knowledge on some of the nodes in the graph, given the homophily relationship between nodes.

Given the bipartite graph and the prior knowledge about devices, we infer the probability of unknown devices being compromised using the guilt-by-association principle. To do so, we model each node $i \in V$ as a random variable, x_i, that can be in the set of *states* $S = \{good, bad\}$ so that the badness and goodness of a node can be expressed by the probabilities $P(Bad)$ and $P(Good)$, respectively, where $P(Bad) + P(Good) = 1$. Our goal is then to determine the probabilities $P(x_i = Good)$ and $P(x_i = Bad)$ for unknown devices by employing three popular graph-inference approaches including (1) Belief Propagation (BP) [30, 55], (2) Label Propagation (LP) [54, 75], and (3) Unsupervised node embedding (Node2Vec) [24] along with downstream supervised learning. Each

[2]For consistency reasons, we follow previous work [12, 55] in choosing "good" and "bad" as BP labels though they may not sound technical.

Table 4. Initial Scores of Nodes and Edge Potentials for BP

(a) Initial scores of nodes

	P(Bad)	P(Good)
Bad	δ	1- δ
Good	1- δ	δ
Unknown	0.5	0.5

(b) Edge potentials for BP

	Bad	Good
Bad	ϵ	$1-\epsilon$
Good	$1-\epsilon$	ϵ

approach outputs a final score for each node based on which we classify devices. In the following, we describe how we apply graph-inference approaches in our context.

Belief Propagation. At each iteration, BP computes i's belief that neighbor j is in state x_j ($m_{ij}(x_j)$), which is defined as:

$$m_{ij}(x_j) = \sum_{x_i \in S} [\phi_i(x_i)\psi_{ij}(x_i, x_j) \prod_{k \in N(i)\setminus j} m_{ki}(x_i)] \tag{1}$$

Concretely, there are three components to compute message $m_{ij}(x_j)$: (1) initial belief score $\phi_i(x_i)$ for i being in state x_i; (2) the product of all messages from i's neighbors excluding j (i.e., i's incoming message vector from $k \in N(i)$); and (3) the edge potential between two neighboring nodes i and j specifying the probability of i being in x_i and j being in x_j.

We assign the initial belief score for each node based on the ground truth labels (Table 4(a)). Also, Table 4(b) represents the edge potential matrix.

Label Propagation. Let Y contain the one-hot encoding vector of the possible states $x_i \in S$. Initially, $Y[i] = 1$ and $Y[j] = 0$ for each labeled node i and unlabeled node j, respectively. At each iteration, LP computes a score for each node based on the random walks to labeled nodes. Formally, we define T_{uu} and T_{ul} as matrix of probabilities to randomly walk from unlabeled to unlabeled and unlabeled to labeled nodes in the graph, respectively. We estimate the probability of each unlabeled node in each state x_i as follows:

$$\hat{Y}_u = (I - T_{uu})^{-1}.T_{ul}.Y_l, \tag{2}$$

where I is the identity matrix, and Y_l is the vector for labeled nodes.

Node2Vec. We model each node $i \in V$ as a d-dimensional feature vector by optimizing a neighborhood preserving objective. We aim to learn a mapping function f from nodes to feature representation. For each node i, we define $N(i) \subset V$ as a network neighborhood of node i generated through a neighborhood sampling strategy. Node2Vec optimizes the following objective function to maximize the log-probability of observing $N(i)$ for each node i:

$$\max_f \sum_{i \in V} log\ Pr(N(i)|f(i)). \tag{3}$$

Learned f is then used as feature vectors for downstream supervised learning algorithms such as Random Forest to train a model; which is in turn used to compute a final score.

4 EXPERIMENTAL RESULTS AND ANALYSIS

4.1 Experimental Setup

Table 5 summarizes the notation for the parameters used.

BP Implementation: We implemented BP in C following the implementation in [35]. It only takes 1.25 seconds to run 10 BP iterations on average for our graphs.

LP Implementation: We implemented LP in [75] and empirically set the decay of the propagation to 0.0001 and the number of iterations to 1,000.

Table 5. List of Parameters for Experiments and Default Values

Notation	Description	Default
N_p	The threshold to define a popular app	1000
vt	The threshold for the number of VT engines detecting the app as malicious	5
τ_{sim}	The threshold for the app traffic similarity	0.0
ϵ	The edge potential parameter in BP	0.51
δ	The initial score parameter	0.99

Node2Vec Implementation: We adapted the implementation[3] [24] to generate features and used Random Forest to train models as we empirically found it outperforms other supervised algorithms including SVM, XGBoost, Ada Boost, Classification Tree, and Logistic Regression. We tune the Node2Vec to use the following configuration: embedding size = 128, walk length = 80, and the number of walks 10. With hyperparameter optimization, we utilize the following parameters with the downstream Random Forest classifier: max_ features = 65, max_depth = 15, n_estimators = 10 and min_sample_split = 2.

Ground Truth Sets: In our bipartite graph, we have two types of nodes: apps and devices. Although graph-inference can be used to classify both apps and devices in principle, we focus on the classification of devices. We thus consider the badness/goodness of apps as unknown. Hence, the inference is driven by two different sets of device ground truth labels: bad devices (D_B) and good devices (D_G). The number of instances in each set is given in Table 3.

Handling Unbalanced Groundtruth: During the graph inference process, each node has two scores representing its badness and goodness. For simplicity, we only mention the badness score in this section. Essentially, at the initial stage, bad devices in the training set have high scores, good devices in the training set have low scores, and devices in the testing set and apps have 0.5 scores, as defined by Table 4(a). Note that the unbalanced initial scores lead to a biased set (good devices in our case) dominating the final scores [6, 30, 32, 35, 40, 53], which is undesirable as our goal is to detect bad devices. Following the previous research using graph inference algorithms in cybersecurity domains [6, 30, 32, 35, 53], we thus downsample good devices for the training set. That is, we randomly choose the equal number of good devices in the training set to the number of bad devices in the training set. Then, we use the remaining devices as testing sets. We perform k fold cross-validations and compute the average of each performance measure.

Cross Validation: We randomly divide each groundtruth set into k folds and run algorithms k times. In each run, we use one fold of bad devices as a testing set for bad devices ($TEST_B$) and the remaining $k-1$ folds as a training set for bad devices ($TRAIN_B$). We downsample $k-1$ folds of good devices as a training set for good devices ($TRAIN_G$) and the remaining devices (D_G - $TRAIN_G$) as a testing set for good devices ($TEST_G$). Then, we run an algorithm to get the final scores of all nodes. Devices in the testing set are classified based on their final scores. Specifically, we vary the threshold for the final score, and classify a device whose final score is above the threshold as bad; otherwise, we classify it as good. We then compute the **true positive rate (TPR)** as the number of bad devices correctly classified to the total number of bad devices in the test set. The **false positive rate (FPR)** is computed as the number of good devices that are misclassified to the total number of good devices in the test set. Algorithm 1 describes the pseudo code of our evaluation process. As we observe varying k has no significant difference, we report the 10-fold classification results.

[3] https://github.com/aditya-grover/node2vec.

ALGORITHM 1: Pseudocode for DeviceWatch Evaluation

D_B: the set of bad devices in Table 3

D_G: the set of good devices in Table 3

$N(TRAIN_B)$: the number of bad devices in the training set

$N(TRAIN_G)$: the number of good devices in the training set

KFoldCrossValidation()

1: Split D_B into k folds, $\bigcup_{j=1}^{k} B_j$
2: Split D_G into k folds, $\bigcup_{j=1}^{k} G_j$
3: **for** $j = 1$ to k **do**
4: Set B_j as a testing set for bad devices, $TEST_B$
5: Set remaining $k-1$ folds of B as a training set for bad devices, $TRAIN_B$
6: Downsample $k-1$ folds of G as a training set for good devices $TRAIN_G$ where $N(TRAIN_B) = N(TRAIN_G)$
7: Set remaining good devices (G - $TRAIN_G$) as a testing set for good devices, $TEST_G$
8: $TRAIN = TRAIN_B + TRAIN_G$
9: $TEST = TEST_B + TEST_G$
10: RunGraphInference($TRAIN$,$TEST$)
11: Compute TPR (Recall), FPR, Precision on the testing set
12: **end for**
13: Average TPR (Recall), FPR, Precision from iteration

RunGraphInference(*TRAIN*, *TEST*)

1: Initialize the scores of devices in $TRAIN$ according to their labels (good/bad) and Table 4(a)
2: Initialize the scores of the rest of the nodes in the graph as 0.5
3: Run the graph inference algorithm
4: Classify the testing devices based on their final score after convergence

4.2 Comparing Graph-Inference Approaches

An important consideration at the early stages of detection is to identify which predictive model would work well to infer compromised devices from graph-structured data. Figure 9 shows the ROC curves for the three graph-inference approaches under consideration (BP, Node2Vec, and LP), where the x-axis represents the FPR and the y-axis represents the TPR. The figure shows that BP provides a low FPR, compared to LP and Node2Vec with RF. A few possible reasons why BP performs slightly better are: (a) Node2Vec fails to capture labels into the embedding and may result in inaccurate classification when two or more nodes have a similar structure but different labels, and (b) LP takes the average of the neighboring node values, and fails to capture the homophily relationships among neighbors. Further, we observe that BP is several orders of magnitude faster than Node2Vec. We thus opt to employ BP to detect compromised devices and focus on analyses drawn from BP in the following.

4.3 DeviceWatch with Belief Propagation

4.3.1 Device Classification Accuracy with Parameter Choices. Following the standard practices on evaluating graph-inference approaches, to measure the performance, we present a series of ROC curves where: the x-axis represents FPR, the y-axis represents TPR, and each point in ROC

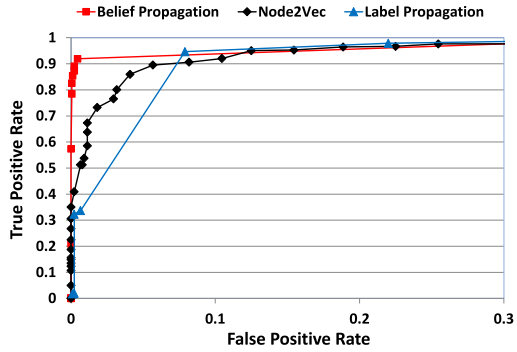
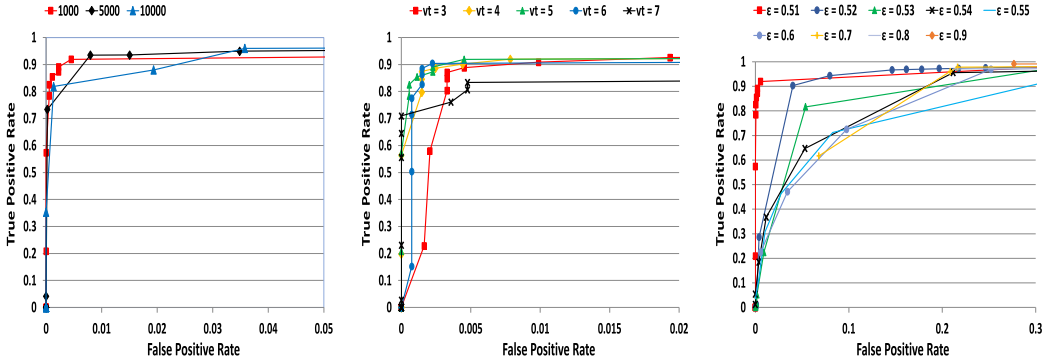


Fig. 9. A ROC curve for three graph-inference algorithms.



(a) Varying popular app definitions ($vt = 5$, $\epsilon = 0.51$)

(b) Various vt ($N_p = 1000$, $\epsilon = 0.51$)

(c) Various ϵ ($N_p = 1000$, $vt = 5$)

Fig. 10. ROC curves with various parameter setting.

curves represents different thresholds for final scores [12, 30, 32, 35, 40, 53, 54]. While doing so, we vary the parameters in Table 5, which might have effects on the performance.

Varying the Definition of Popular Apps. Figure 10(a) shows the ROC curves with varying the definition of popular apps. Although not significant, the figure shows that with less filtering such as $N_P = 10000$, FPR increases for a fixed TPR. This is expected because many devices, if not all, using popular apps will be considered related, resulting in false associations [6, 30, 35]. Recall that only 1.4% of apps are filtered with $N_P = 1000$ (Figure 7). This means that filtering popular apps is helpful in avoiding false associations, yet does not have a notable impact on the total number of apps in the graph.

Varying vt Values. Note that there is no consensus on the right value for vt [27, 66]. We thus show the results with various vt, i.e., $vt = 3, 4, 5, 6, 7$ in Figure 10(b). As the figure shows, there is no significant difference on the FPR and TPR while changing vt, for $vt \geq 5$. We thus use $vt = 5$ for further discussion in the following.

Varying Edge Potentials ϵ. Figure 10(c) shows the ROC curves while varying ϵ. Concretely, when $\epsilon = 0.51$, we achieve 0.92 TPR with only 0.004 FPR. Interestingly (as it is different from BP behavior in other applications such as [30, 35]), our results are sensitive to ϵ, an edge potential parameter of BP. As shown in Figure 10(c), as ϵ increases, FPR increases. Furthermore, we achieve the best accuracy with low values of ϵ (e.g., 0.51).

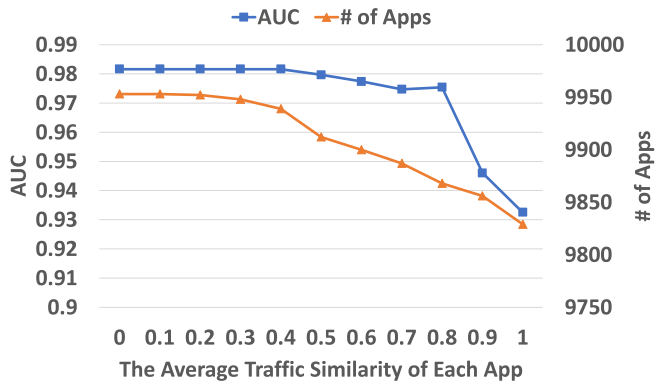


Fig. 11. AUC vs the number of apps filtered with varying τ_{sim} ($vt = 5$, $\epsilon = 0.51$).

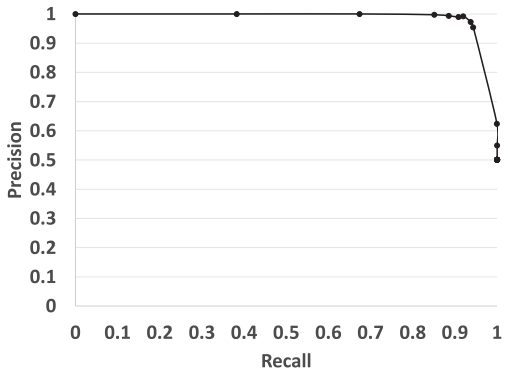
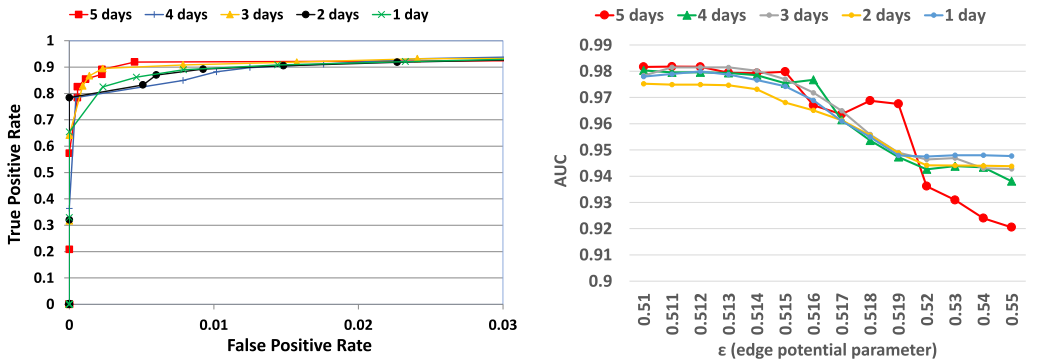


Fig. 12. The Precision-Recall curve($vt = 5$, $\epsilon = 0.51$, $N_p = 1000$).

Varying App Traffic Similarities τ_{sim}. While the low traffic similarity does not mean that apps are not correctly identified [2, 61], an administrator may further filter apps for better confidence on the identified apps. On the other hand, filtering a lot of apps and thus filtering a large number of devices may increase false negatives. We thus show the trade-off between the area under ROC curve (AUC) and the number of apps in the graph when varying τ_{sim} in Figure 11, where the x-axis represents the average traffic similarity of each app, the left y-axis represents the AUC, and the right y-axis represents the number of apps in the graph. As shown in the figure, the AUC greatly decreases, when $\tau_{sim} = 0.8$ and above, where almost 100 apps are filtered. In previous research on app identification, it is shown that the traffic of apps can be correctly classified using τ_{sim} between 0.5 and 0.9 [57, 61]. According to our results along with previous research, to achieve a good balance between app identification, AUC, and the number of apps, a network administrator may choose τ_{sim} between 0.5 and 0.8.

Precision and Recall Curve. While ROC curves provide us to help find an appropriate TPR at an acceptable FPR, it is also important to measure precision and recall. We thus further present the Precision-Recall curve in Figure 12. As shown in the figure, we can achieve a good balance between precision and recall where we achieve 0.992 precision and 0.92 recall at best.

4.3.2 Device Classification Accuracy with Different Time Periods. To see if different behavior of BP is caused by a specific data point or unique characteristics of mobile networks, we perform experiments with varying the time range of our dataset. Specifically, we divide our 5 days dataset



(a) ROC curve for different days dataset ($vt = 5$, $\epsilon = 0.51$) (b) AUC when varying ϵ for different days dataset ($vt = 5$, $N_p = 1000$)

Fig. 13. ROC curves and AUC drawn from various time period setting.

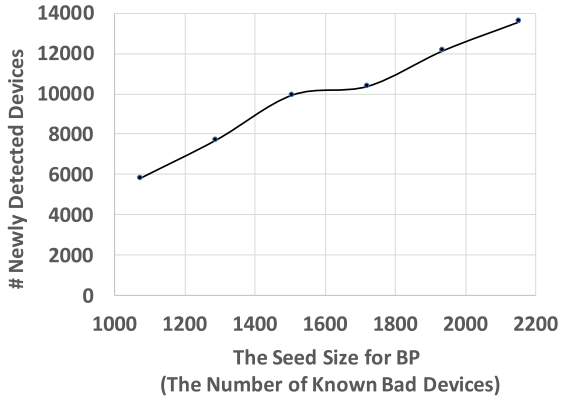


Fig. 14. The number of newly detected devices when varying the seed size.

into (1) 1-day and 4-days datasets and (2) 2-days and 3-days datasets. Figure 13(a) shows that the ROC curves drawn from various time periods with $\epsilon = 0.51$ and Figure 13(b) shows the area under the ROC curve (AUC) drawn from various time periods with various ϵ. Figures 13(a) and 13(b) show that there is no significant difference between different time ranges on the best FPR/TPR and the effect of edge potential parameter. In other words, our analysis generally holds in mobile networks, not for a specific data point.

In Section 4.4, we present an in-depth analysis to demonstrate distinctive characteristics of the mobile traffic data that lead to this behavior.

4.3.3 Device Classification Coverage. While we showed that our approach can achieve low FPR and high TPR in detecting devices, it is also important to show how many unknown bad devices can be detected beyond those in the seeds (i.e., known bad devices). Essentially, a naive baseline approach of utilizing a blacklist is only detecting devices having previously known bad apps (i.e., those in our ground truth). However, our approach, in general, can expand the limited knowledge greatly so that we detect $6 \sim 7$ times as many unknown bad devices not in the ground truth as shown in Figure 14. In Section 5, we show that these newly detected devices indeed show risky behavior.

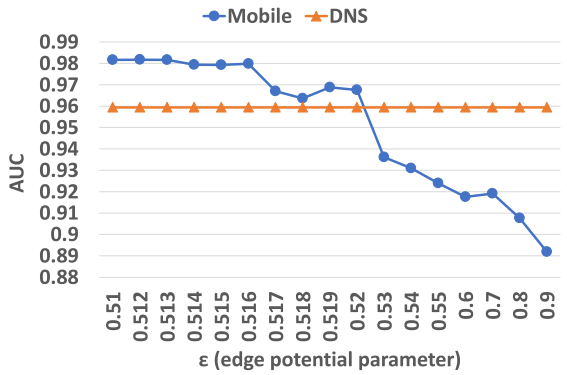


Fig. 15. AUC when varying ϵ for Mobile and DNS.

4.4 Measurements on the Unique Characteristics of Mobile Network Structure and In-Depth Analysis of BP Behavior

Essentially, the key to the effectiveness of graph inference-based approaches is how the graph is constructed (i.e., the graph structure). Specifically, the approach will work well depending on (1) the well-defined association while reflecting homophily relationships and (2) how the prior knowledge on a few nodes will be propagated to classify others. In Section 4.3, we show that we can successfully identify bad devices using the graph-inference over the defined device-app association. In this section, we will study the unique graph structure of mobile networks (i.e., bad devices are close to each other compared to others yet all nodes in the graph are also relatively highly connected with each other) resulting in that BP works best with carefully chosen parameters. In such a highly connected graph structure, Node2Vec and LP fail to distinguish bad and good devices as mentioned in Section 4.2. We thus focus on describing how the device-app graph structure affects the behavior of BP.

Identifying the most effective ϵ values to accurately classify graph nodes using BP is a known problem, and recent work aims to automate the process of identifying such values [72]. It has been observed in previous work [30, 35] that the accuracy of BP is not significantly impacted by different ϵ values, whereas our results are sensitive to ϵ (Figure 10(c)). In the following, we shed light on why ϵ has an impact on our results, by providing in-depth analysis on distinctive network properties of two bipartite graphs from different applications: *Mobile* and *DNS*. By doing so, we explain how to choose the optimal parameters for DeviceWatch with BP to achieve the best accuracy.

Mobile represents the bipartite graph built from our dataset. Note that various vts do not have a significant impact on the accuracy and topological locations of nodes in the ground truth. We thus use the ground truth with $vt = 5$ to provide analysis in the following.

DNS represents the bipartite graph between domains and IPs built from the active DNS dataset in [30]. In [30], Issa et al. applied BP on domain-hosting IP bipartite graph to detect malicious domains. To compare the impact of ϵ in different networks, we obtained the domain-IP bipartite graph and the ground truth labels on domains used in their work [30].

To clearly capture the sensitivity to ϵ in each of the two graphs (*Mobile* and *DNS*), we measure the area under the ROC curve (AUC). Figure 15 shows AUC (the y-axis) with varying ϵ (the x-axis). The figure clearly shows that ϵ has an obvious impact on AUC in *Mobile*, whereas AUC in *DNS* stays almost the same (0.96) regardless of ϵ. We argue that this different behavior of BP is due to the network structures and the topological locations of nodes in the ground truth.

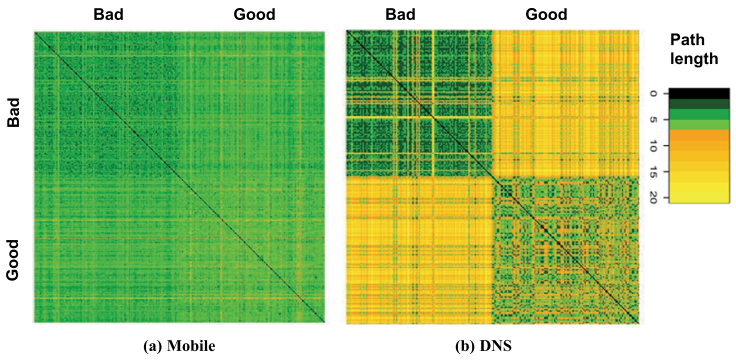


Fig. 16. Shortest path length matrix between bad/good and bad/good nodes.

For any two nodes S and T in the graph, their impact on each other depends on multiple variables, the most important of which are: (1) **the length of the path** between S and T, (2) **the number of paths** between S and T, and (3) **the edge potential parameter**.

First, the longer the path between S and T, the smaller S's impact on T. This is because the edge potential diminishes as it travels on the path between the two nodes (due to fraction multiplications as many as the length of the path.) As a result, the final badness score will be insensitive to ϵ in the case of graphs with longer paths. Second, the larger the number of paths between S and T, the higher the impact of S on T. This is because the final score at T is a function of the product of messages received on each path from S to T. For example, assume that a bad node S has p paths to T, then S sends a bad message $m_B(i)$ and a good message $m_G(i)$ on a path i. Since S is bad, $m_B(i)$ is larger than $m_G(i)$. The final bad (good) impact of S on T is a function of the product of the $m_B(i)$ ($m_G(i)$) messages from all the p paths. The larger the number of paths (p), the higher the difference between the $m_B(i)$ product and the $m_G(i)$ product, and hence, the higher the final badness score (due to the assumption that S is bad in the example). Finally, if we set $\epsilon = 1$, the path length will no longer have any impact, because length-1 has the same impact as length-1000; if we set ϵ close to 0.5, b_S's impact on b_T greatly diminishes except for very short paths (e.g., 2).

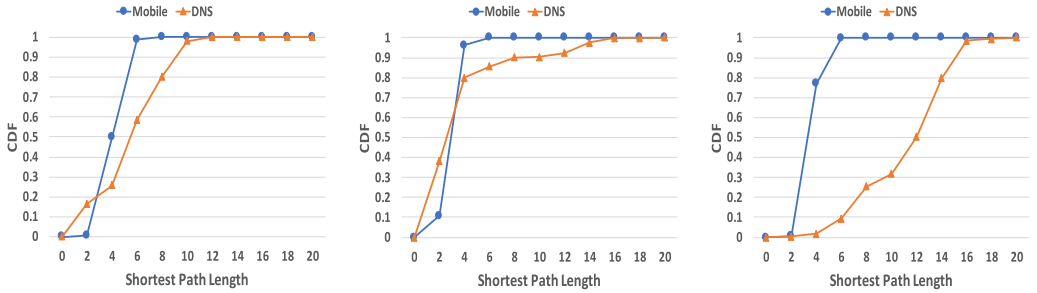
We next compare two graphs (*Mobile* and *DNS*) in terms of their topological features that have an impact on BP as explained earlier. Specifically, we are interested in nodes in the ground truth set.

Shortest path length. Consider two clusters: bad (C_B) and good (C_G). The important intuition behind BP using the homophily relationship is that each cluster's intra-cluster distance is low, whereas inter-cluster distance between two clusters is high. We thus measure the intra-cluster and inter-cluster distances in terms of the shortest path lengths between all pairs of nodes in C_B and C_G.

Figure 16 shows the matrix representing the shortest path lengths between nodes in C_B and C_G. The range of lengths is from 0 (black) to 20 (yellow). The figure shows a few important observations. First, in both graphs, intra-cluster distances are smaller (darker and greener) than inter-cluster distances between C_B and C_G. Second, C_B's intra-cluster distances are the lowest (darker and greener) in both graphs. Finally, the difference between intra-cluster and inter-cluster distances in *DNS* is much larger than that in *Mobile*.

Figure 17 presents the CDF (the y-axis) of shortest path lengths between nodes in C_B and C_G (the x-axis); one line per dataset.

Interestingly, the difference of C_B's intra-cluster distances between *Mobile* and *DNS* is relatively small. On the other hand, we observe different characteristics in C_G's intra-cluster distances, and inter-cluster distance between C_B and C_G for each dataset. In *Mobile*, 98.7% of path lengths between



(a) CDF between good nodes (b) CDF between bad nodes (c) CDF between bad and good nodes

Fig. 17. Shortest path length CDF between bad/good and bad/good nodes.

Table 6. Network Properties of the Mobile and DNS Datasets

	Mobile CC	**DNS CC**	**Mobile EC**	**DNS EC**
Bad	0.229	0.088	0.03	0.005
Good	0.204	0.141	0.01	0.179
Unknown	0.212	0113	0.021	0.006

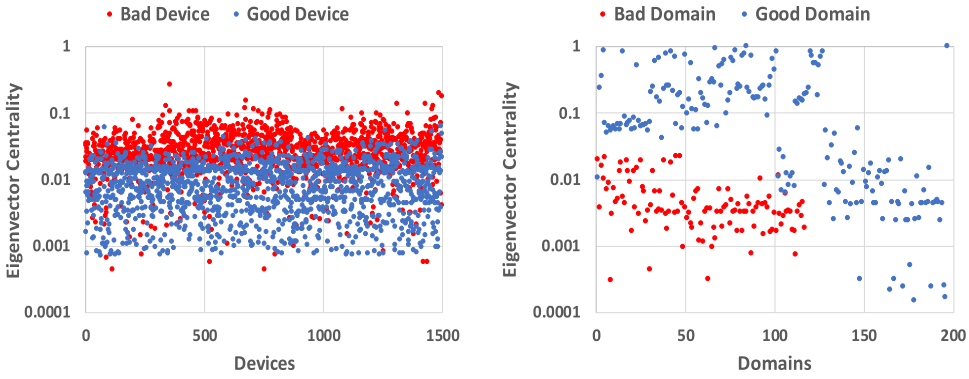
nodes in C_G are smaller than or equal to 6, which is in fact similar to the inter-cluster distance between C_B and C_G where 99.8% of path lengths are smaller than or equal to 6. In *DNS*, 60% of path lengths between nodes in C_G are smaller than or equal to 6, while 90% of path lengths between nodes in C_B and C_G are more than 6. In other words, although the intra-cluster distance is smaller than the inter-cluster distance in both datasets (i.e., the homophily relationships hold), the difference between intra-cluster and inter-cluster distances in *Mobile* is relatively small, whereas the difference is relatively large in *DNS*.

Observation 1. *Relatively long inter-cluster distance diminishes the impact of bad (good) domains on good (bad) domains, irrespective of ϵ in* DNS*; whereas ϵ plays a big role in classification accuracy in* Mobile*, due to the small differences between intra-cluster and inter-cluster distances. Concretely, bad devices have more impact on good ones when we use higher ϵ, resulting in higher false positives. Hence, it is required to carefully choose ϵ close to 0.5 (e.g., 0.51) to avoid high false positives.*

Closeness centrality. (CC) of a node measures the average length of the shortest paths from the node to others [56]. Essentially, CC takes into account both factors: the number of paths and the shortest path lengths. If all nodes in the graph are highly connected to each other with short path lengths, the CCs of all nodes will be similar. Indeed, the average CCs of bad and good devices in *Mobile* are similar; whereas, the average CC of bad domains is relatively small, compared to those of good and unknown domains in *DNS* (Table 6).

Observation 2. *Along with the Observation 1, we can conclude that the bad nodes in* DNS *are much farther from other nodes and have less number of paths to other nodes, while good nodes are highly connected to good or unknown nodes. This is expected as good domains are not likely to have many connections to bad domains, but have many connections to good or unknown domains. Hence, the classification accuracy is not sensitive to ϵ in* DNS.

Eigenvector centrality. (EC) of a node measures its influence in the graph. A node with high EC means that it is highly connected to other *influential* nodes. That is, messages are most



(a) Mobile EC distribution (b) DNS EC distribution

Fig. 18. Eigenvector centrality distributions of the Mobile and DNS.

frequently passing through a node with high EC so that it will play a key role during BP process. As shown in Table 6, there is a clear difference on ECs between *Mobile* and *DNS* graphs.

Observation 3. *The average ECs of bad, good, and unknown devices are almost similar in* Mobile. *This means that all nodes in the graph are highly connected with each other so that no significantly influential nodes are in the graph. Note that the ECs of bad devices are the highest, meaning that as the higher ϵ is used, the score of bad devices can dominate the network, resulting in high false positives. On the other hand, the average EC of good domains is much higher than those of bad and unknown domains in* DNS.

Figure 18 shows the distribution of ECs of bad and good nodes in each dataset. Similar to results from *CC*, Figure 18(b) shows that bad domains in *DNS* are significantly further from other nodes and are not connected to influential nodes, meaning that there is a smaller number of paths to other nodes. Although the *EC*s of good domains are high on average, they are well-distributed. This is in fact expected, as there can be influential and non-influential domains.

By our definition in *Mobile*, bad devices can have edges with all types of apps (i.e., bad, good, suspicious, and no-info apps); good devices can have edges with good and no-info apps. This means that good devices could have a similar number of paths with both good and bad devices; bad devices, however, have more paths with other bad devices than good devices. Consequently, bad devices become relatively influential and connected to other influential bad devices, resulting in the relatively high *EC*s as shown in Figure 18(a).

Observation 4. *Bad devices are more influential on others than good devices in* Mobile*; whereas bad domains are less influential on others in* DNS. *Along with the Observation 1, we can conclude that bad devices get more influences from bad devices, especially from those influential bad devices, than good devices; so that good devices' messages have relatively less impact on bad devices. Consequently, there is not much change on false negatives, irrespective of ϵ, as opposed to false positives.*

Summary. In short, we show that bad devices are close to each other compared to good devices in the device-app association graph (i.e., the homophily relationship holds), which makes us successfully identify bad devices using the graph inference. On the other hand, we also show that the graph structure of the device-app graph is unique in that the length of paths between good and bad devices is relatively shorter and the number of paths between them is relatively larger, compared to DNS graph. Considering these characteristics, we thus suggest choosing ϵ close to 0.5 (e.g., 0.51) to achieve the lower false positives with DeviceWatch.

5 POST ANALYSIS OF NEWLY DETECTED DEVICES

To further verify the correctness of the classification results in Section 4.3.1, we study how risky the behavior of newly detected bad devices is, compared to good devices. Specifically, we compare newly detected devices against good devices in terms of private information leakage (Section 5.1) and underlying network infrastructure accessed by the devices (Section 5.2). To do so, we examine samples of classified good and bad devices. Concretely, we sort unknown devices by their final scores, and choose top-100 devices with high scores as bad and bottom-100 devices with low scores as good.[4]

5.1 Leaking Private Information

Devices having bad apps are known to often leak private information [14, 34]. Note that non-bad apps, and thus good devices having them, may also leak information [20, 50]. However, in the following, we show that the behavior of bad devices is riskier than good devices in terms of leaked information type and leakage traffic statistics.

Ethics. It is important to note that our research is conducted on an anonymized version of the dataset where possible privacy concerns are carefully considered and addressed. Packet payloads are encrypted in the traffic and HTTP headers for part of the traffic are available. Information in HTTP headers is often in a structured format, i.e., a key-value pair [26, 49, 50]. Other than the fields mentioned in Section 3.1 (i.e., those for identifying apps and devices), ISP securely replaced all values (e.g., phone numbers, user names, device identifiers) appearing in the traffic with random strings before they provide the data to authors. Hence, we could only know the key, not the values. This anonymization process ensures that the authors could not link the traffic to any individual user. To further secure the data, the anonymized dataset is kept in a dedicated server where only one authorized author has access.

Privacy leaks often occur with a specific key-value pair [26, 49, 50]. For example, a login password leaks with *pwd=mypaSS123* where *pwd* is a key and *mypaSS123* is a value. Note that the key for a specific type of private information might be different depending on each app or device [49, 50].

We have compiled a list of private information which often leaks in mobile networks based on the previous research [20, 49, 50, 69]. As the leakage occurs only in the HTTP traffic, we inspect HTTP packets originated from the device to analyze the privacy leakage. Then, from the dataset, we heuristically extract highly-related keywords to each type of private information, examples of which are summarized in Table 7. Note that we only present a few examples of keywords, as the keywords are similar yet small variations such as imei1 and imei7. While the private information might have been obfuscated in the traffic (e.g., hashing), it is also shown that most information leakage occurs in plaintext in mobile networks [50]. We thus argue that the following results represent the general behavior of each device.

We search the keywords in each device's all of the HTTP headers, and consider a device leaks its private information if a non-empty key-value string corresponding to given keywords is found in any headers. These leaking packets may or may not include the app string. While we analyze all leaking packets from a device, we further provide analysis on leaking packets having the identified app. We consider an app as the leaking app if we can identify the app from the leaking traffic with our approach in Section 3. For example, let us assume device1 has five packets as shown in Figure 19. The device1 has four leaking packets and it has two leaking apps. We found 69 leaking apps on 200 devices including bad and good devices.

[4]Note that we choose scores derived from the results with parameters $vt = 5$ and $\epsilon = 0.51$, based on the discussion in Sections 4.3.1 and 4.4.

Table 7. Example Keywords to Extract Private Information

Category	Type	Example Keywords
Phone	Sim card number	iccid, simserialnumber, simno, simnumber, sim
	IMSI	imsi, mobileimsi, user-imsi, imsi1, imsi_no, x-imsi, client-imsi
	Phone number	phone_num, phone, tel_num, mobile_no, cellphonenumber, cellphone, userphone, tel_number, usrphonenum
User	User ID	user_nick, user_id, user_name, userid, x-userid, log-user-id, login_name, client-user-id
	Email	user_email, email, login_email, email_name, contact_email, acc_email
	Birthday	birthday, user_birth, customer_birthday, passenger_birthday
	Gender	gender, sex, client_gender
Credential	Password	userpwd, password, passwd, user_pwd, _password, pwd
Location	Location	longitude, latitude, lng_lat, coordinate, homeaddress, coords, geo_location, gps_long, geoInfo
Device Identifier	UDID	udid, device_udid
	Device ID	devid, device_id, x-device-id, deviceid
	IMEI	imei, device-imei, phone-imei, imei1, mobileimei
	GUID	guid, phoneguid, dev-guid
	UUID	uuid, device_uuid, phoneuuid, x-device-uuid, uuid2
	MAC	user_mac, mac, mac_addr, _mac, x-macaddress
	Android ID	android_id, androidid, _androidid, androidid1

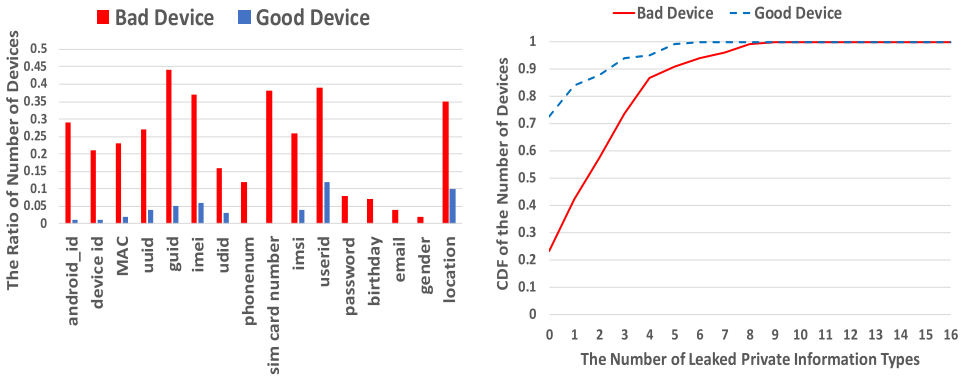
Devices	Leaked Private Information	App String
device1	location	-
	deviceid	com.my.app
	email	-
	gender	com.my.app2
	-	com.my.app3

Fig. 19. Example traffic originated from a device.

Figure 20(a) presents the ratio of the number of devices (the y-axis) leaking each type of private information (the x-axis). Generally, a large number of bad devices leak private information compared to good devices. Notably, only bad devices leak highly sensitive information including phone/sim card numbers, passwords, birthdays, email, and genders. Interestingly, some of the good devices also leak private information.[5] One reason might be that even non-bad apps, particularly location-based searching apps, sometimes leak information such as location and device identifiers in order to support their functionality [20, 50]. In fact, we observe that most of the leaking apps on good devices are location-based searching apps.

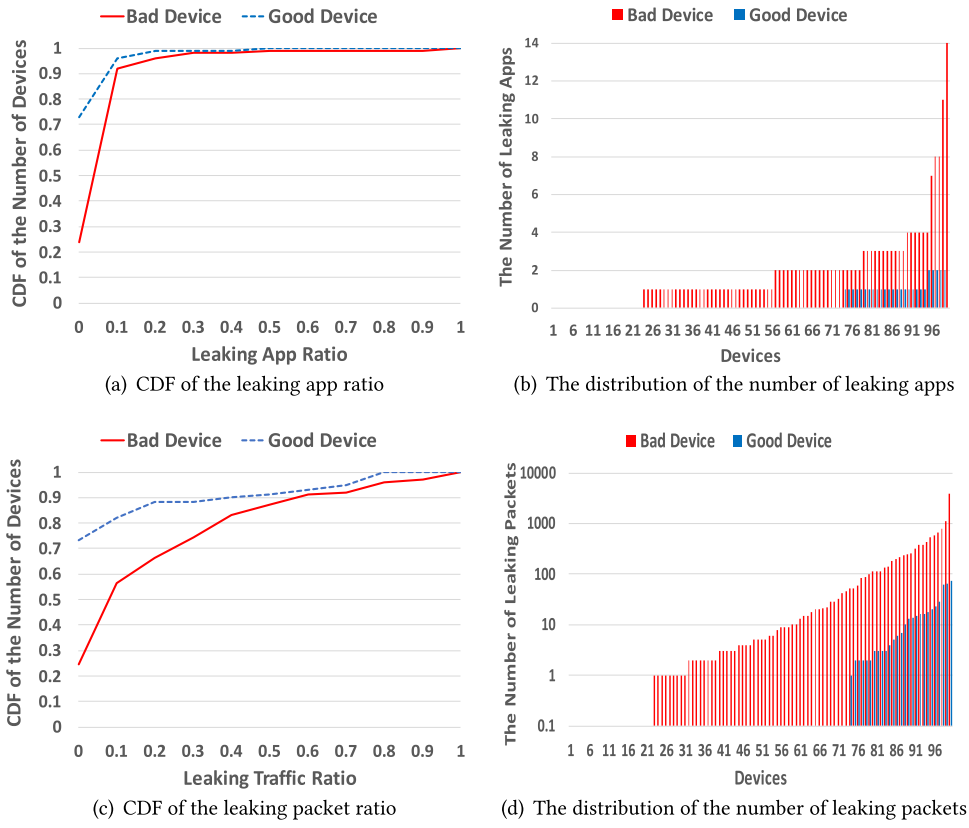
To compare leaking on good and bad devices, we measure (i) the number of leaked private information types, and (ii) the distribution of the number of leaking apps and packets. Figure 20(b) shows the CDF (the y-axis) of the number of leaked private information type of devices (the x-axis). As the figure shows, 73% of good devices do not leak any information and 10% of good devices leak

[5]Note that, as we select good devices with bottom-100 scores, it is less likely that these classified good devices include false negatives.



(a) Statistics for private information leakage (b) The CDF of the number of leaked information types

Fig. 20. Private information leakage.



(a) CDF of the leaking app ratio (b) The distribution of the number of leaking apps

(c) CDF of the leaking packet ratio (d) The distribution of the number of leaking packets

Fig. 21. The distribution of leaking apps and packets of devices.

only one information type; whereas 77% of bad devices leak at least one information type and 10% of bad devices leak more than five information types.

We further measure how many apps and packets of each device leak private information. Figures 21(a) and 21(c) present the CDFs of the leaking app and the leaking traffic ratios,

Table 8. The Number of Leaked Information Types in ISP Traffic, in Manually Captured Traffic, and in Both Traffic

App	ISP (only in the ISP traffic)	The captured traffic (Only in the captured traffic)	In both traffic
News app1	4 (1)	5 (2)	3
News app2	4 (1)	3 (0)	3
Book reading app	8 (3)	8 (3)	5
Game app	3 (1)	2 (0)	2
TV platform app	9 (3)	9 (3)	6

respectively. The x-axis in each figure represents the leaking app and leaking traffic ratios, respectively; the y-axis represents the portion of devices.

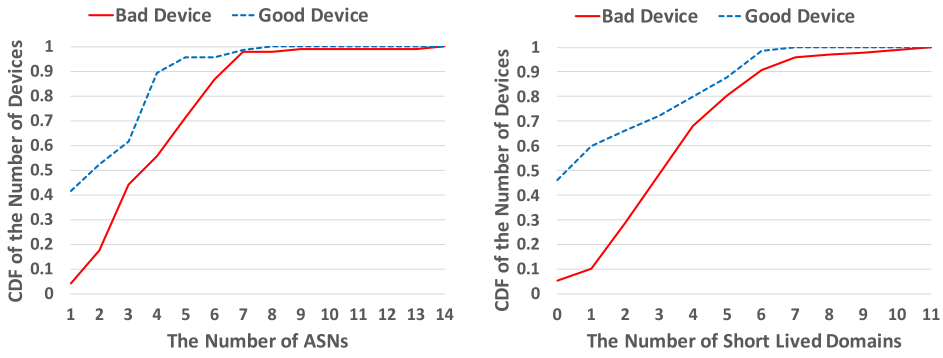
The leaking app ratio is measured by the number of apps leaking information over the total number of apps of each device; the leaking traffic ratio is measured by the number of packets leaking information over the total number of packets of each device. As shown in the figures, although some good devices also leak private information, the ratios of leaking apps and packets of each device are relatively small. Specifically, we observe that among all the good devices that leak private information (i.e., 27% of the good devices), 85% of them have less than 10% of leaking apps (Figure 21(a)); the traffic of half of them has less than 20% of leaking packets (Figure 21(c)).

Figures 21(b) and 21(d) present the distribution of the number of leaking apps and packets of devices, respectively. The x-axis represents each device; the y-axis in each figure represents the corresponding number of leaking apps and packets of each device, respectively. Note that 73% of good devices and 24% of bad devices do not leak any information so that their numbers of leaking apps and packets are 0, which are not shown in the figures.

Interestingly, these leaking apps are not necessarily in the ground truth set of bad apps in Section 3. In fact, 81% of these leaking apps (i.e., 56 out of 69 leaking apps) are not the apps originally flagged by VT. In other words, the inference relies on a largely independent set of apps compared to the ground truth to detect bad devices.

Recall that we do not have physical access to devices in the traffic, and thus we cannot perform comprehensive measurements about other kinds of possible privacy leakage on each individual device in the traffic. To justify our findings in this section, we employ an alternative approach. As discussed in Section 3, we attempt to download app binaries by searching the app string from multiple app stores. However, some app strings are not searchable [39, 48], as mentioned in Section 3.1 and we only have two Android devices and no iOS devices for manual inspection. Out of 69 leaking apps, we are able to find five app binaries to be installed in two devices mentioned in Section 3 for manual verification. We run these five apps on the two devices, capture the network traffic, and perform the same analysis. Specifically, we search the keywords in Table 7 from the captured traffic, check if there is privacy leakage in the traffic, and check if the corresponding type of information to the keywords is actually leaked. Then, we compared the difference in the number of leaked information types between the ISP traffic of compromised devices and the captured traffic in Table 8. By doing so, we indirectly show our leakage analysis based on keyword searching in fact reflects the actual leakage. We observe that there is a large overlap between the ISP traffic and the captured traffic (3rd column in Table 8). One possible reason for the difference between the set of keywords is that we captured traffic on Android devices, but the ISP traffic includes iOS devices. For example, "UDID" is iOS-specific device information revealed in the ISP traffic.

Assumption and Limitation in Leaking Information Analysis. Note that we are not able to directly verify the leakage in each device due to our data limitation. We assume the non-empty



(a) the number of ASes (b) the number of short lived domains

Fig. 22. The CDFs of the number of ASes utilized and the number of short lived domains accessed.

string in matching keywords as leaked information, but it is also possible that the value is obfuscated. To have better confidence on our analysis and justify our findings, we perform case studies as an alternative approach. We are able to find 5 out of 69 apps to manually verify our findings using the alternative approach. We thus do not claim that our analysis covers all possible scenarios of leaking information. However, we argue that our analysis still suggests a clear difference between detected bad devices and good devices to a certain extent in terms of leaking information.

Summary. In short, we show that although devices do not have bad apps from the ground truth, classified bad devices are showing undesirable behavior in terms of privacy leakage. It is important to note that highly sensitive information such as passwords is only leaked on bad devices. This result is also promising in that we can possibly identify unknown bad apps by our approach. Evidently, apps leaking highly sensitive information are not desirable and we may further analyze apps causing privacy leakage on the classified bad device. Since our focus is to identify devices, we leave further investigation on apps as future work.

5.2 Network Infrastructure Accessed

We analyze the domains and IPs accessed by classified devices. It is well-known that miscreants utilize fast fluxing [25] to improve the availability of their malicious domains. Further, miscreants move their domains from one hosting provider to another frequently to evade take down [30]. In our dataset, 94% of hosting providers possess a single **AS (Autonomous System)**. Thus, the above observation on hosting providers can be generalized to ASes. We seek to find out if indeed this AS behavior exists in the IPs accessed by the apps in the classified devices. Figure 22(a) shows the CDF of the number of ASes that host domains accessed by classified good and bad devices. Consistent with findings from previous research, it shows that bad devices tend to access IPs from more ASes compared to good devices: 60% of bad devices access IPs from more than three ASes, whereas only 40% of good devices exhibit the same behavior during the study period. As it is economical to create domains, nowadays, miscreants use many disposable domains to launch their attacks [65]. We thus also explore whether domains accessed by bad devices exhibit such behavior. First, for the domains in the dataset, we extract the first seen and last seen dates of each domain from Farsight passive DNS repository [22], which collects DNS queries resolved world-wide and serves historical DNS query data since 2011. We define a short lived domain as one whose DNS footprint is less than three months. Most of these domains are usually taken down, sink holed, or black listed, if identified as malicious [4]. Figure 22(b) shows the CDF of the number of short lived domains accessed by good and bad devices. The figure confirms with the previous research findings where,

in general, classified bad devices access more short lived domains compared to good devices. It shows that 60% of bad devices access more than three short lived domains, whereas nearly half of good devices never access short lived domains during the study period.

6 DISCUSSION AND LIMITATIONS

Potential Evasion on Device and App Identification. Our approach may not cover all the devices and apps in the traffic. One may use evasion techniques using NAT IPs, obfuscated app strings, and VPNs. While in Section 3.1, we showed that those are not likely to be the case in our traffic, our approach may encounter additional challenges when such obfuscated techniques are prevalent. Note that identifying devices and apps from heterogeneous traffic is still in its early stage in the literature [33, 57, 58, 61], and key assumptions for previous approaches do not apply to our data in which multiple devices run multiple apps simultaneously. We leave the investigation of this direction as future work.

Ground truth and VT intelligence. We establish ground truth by querying the app strings to VT intelligence. An organization may build ground truth using alternative approaches (e.g., querying domains of each certificate to VT, manually inspecting a small number of devices in the network, and utilizing other public sources such as [1, 3]. We argue that the source of establishing ground truth is independent of our approach. Our contribution is to identify unknown devices that may have unknown bad apps by expanding the limited knowledge based on any accessible intelligence source through graph inference. In fact, it is the key advantage of our approach: detection is based solely on device-app association and we need neither deep inspection on devices and apps nor the set of specific features such as device models or app types. As discussed in Section 5, although our groundtruth is built by using VT, we show that detected devices show other types of risky behavior such as leaking highly sensitive information (e.g., passwords) or accessing risky network infrastructure. 81% of apps on such devices are not detected by VT.

Homophily Relationships: Device-App Association vs Device-IP Association. Essentially, the key to the effectiveness of graph inference-based approaches is if the defined association in fact has homophily properties. We empirically show that our proposed device-app association has such a property, and thus we can successfully identify compromised devices by applying graph-inference over the device-app association. We also show that bad devices are close to each other compared to good devices in the device-app graph (Section 4.4). To further justify the homophily property of device-app association, we compare with an alternative approach, i.e., using the *device-IP association*. Intuitively, if a device is heavily communicating with a malicious destination IP, the device is compromised and it is likely to connect to other malicious IPs. However, a single IP does not correctly represent a user's general behavior, while most activities on mobile devices incur through apps and the same IP may host multiple unrelated apps. Hence, if we use device-IP association, it considers all devices communicating with the same IP as related, resulting in many false associations leading to poor results (i.e., no homophily properties). Figure 23 shows the ROC curve when using device-IP association based inference. The figure clearly shows that the performance of the classification based on the device-IP association is not acceptable in practice as it misclassifies a considerable number of devices.

Privacy Implication on Traffic Monitoring. While traffic monitoring at an ISP level can provide a more centralized and efficient way to guarantee security, it may open the door for censorship, user profiling, and user tracking. However, we want to emphasize that our goal is not to pinpoint the exact app of each device, yet to identify compromised devices. When DeviceWatch is deployed at the ISP level, privacy concerns should be carefully considered. Additional studies are required to address the possible privacy implications, we leave as a future direction. We describe some

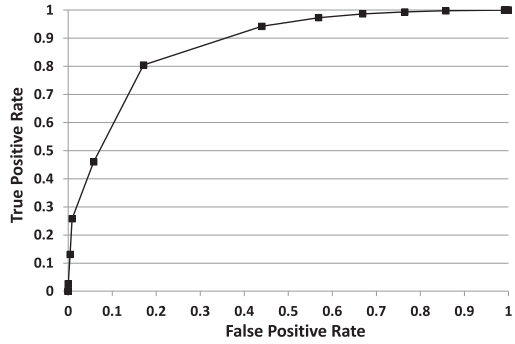


Fig. 23. ROC curve with device-IP association based inference ($vt = 2$, $\epsilon = 0.51$, $N_p = 1000$).

recommendations to prevent privacy issues. First, the network administrator needs to carefully consider privacy implications. For example, all app names excluding malicious ones should be replaced with pseudo app names so that sensitive information cannot be revealed from the usage of apps (e.g., medical apps). In Section 3.1, we propose to use traffic similarity measure τ_{sim} to control the trade-off between the accuracy of app identification and the coverage of apps. This parameter can also be used to control the trade-off between the accuracy of app identification and privacy. For a better privacy guarantee, ISP may choose to use low τ_{sim} (e.g., 0.5) while still ensuring confidence in app identification. App developers should obey privacy guidelines, properly encrypt their traffic, and obfuscate traffics to prevent posing any threats to user privacy. Users also need to be aware of potential privacy threats and consider extra security measures such as using VPNs.

7 CONCLUSION

We proposed a data-driven network analysis approach with graph-inference to identify compromised mobile devices. In doing so, we define a device-app association based on the intuition that devices sharing a similar set of apps will have a similar probability of being compromised (i.e., homophily relationships). We studied this problem on real-world data that faithfully represents the actual behavior of mobile users with which we demonstrate the effectiveness of our approach. We empirically show that homophily relationships exist by showing that bad devices are close to each other in the device-app graph and we can thus successfully identify compromised devices using the graph-inference over the device-app association. We further study the impact of graph topology on BP parameters and highlight the distinct features of the mobile graph. Finally, our privacy leakage and hosting infrastructure analyses support the claim that our approach can reliably detect unknown compromised devices without relying on device-specific features. It is also important to take appropriate actions after compromised devices are detected. We also discuss that further investigation on detected devices might be helpful to identify unknown malicious apps, which we leave as future work.

REFERENCES

[1] 2019. Koodous: Online malware analysis platform. https://koodous.com/.

[2] Hasan Faik Alan and Jasleen Kaur. 2016. Can Android applications be identified using only TCP/IP headers of their launch time traffic? In *ACM Conference on Security & Privacy in Wireless and Mobile Networks*. 61–66.

[3] Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2016. AndroZoo: Collecting millions of Android apps for the research community. In *MSR'16* (Austin, Texas). ACM, New York, NY, USA, 468–471.

[4] Eihal Alowaisheq, Peng Wang, Sumayah A. Alrwais, Xiaojing Liao, XiaoFeng Wang, Tasneem Alowaisheq, Xianghang Mi, Siyuan Tang, and Baojun Liu. 2019. Cracking the wall of confinement: Understanding and analyzing malicious domain take-downs. In *NDSS*.

[5] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. 2014. DREBIN: Effective and explainable detection of Android malware in your pocket. In *NDSS*, Vol. 14. 23–26.

[6] Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. 2011. EXPOSURE: Finding malicious domains using passive DNS analysis. In *NDSS*. 1–17.

[7] Chad Brubaker. 2018. Protecting users with TLS by default in Android P. https://android-developers.googleblog.com/2018/04/protecting-users-with-tls-by-default-in.html.

[8] Chad Brubaker. 2019. An Update on Android TLS Adoption. https://security.googleblog.com/2019/12/an-update-on-android-tls-adoption.html.

[9] Graeme Burton. 2017. Australia wants to force ISPs to protect customers from malware. https://www.theinquirer.net/inquirer/news/3009045/australian-wants-to-force-isps-to-protect-customers-from-malware.

[10] Fangda Cai, Hao Chen, Yuanyi Wu, and Yuan Zhang. 2015. AppCracker: Widespread vulnerabilities in user and session authentication in mobile apps. *MoST* (2015).

[11] Frank Cangialosi, Taejoong Chung, David Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, and Christo Wilson. 2016. Measurement and analysis of private key sharing in the https ecosystem. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 628–640.

[12] Duen Horng Chau, Carey Nachenberg, Jeffrey Wilhelm, Adam Wright, and Christos Faloutsos. 2011. Polonium: Tera-scale graph mining and inference for malware detection. In *SDM*. SIAM, 131–142.

[13] Kai Chen, Peng Wang, Yeonjoon Lee, XiaoFeng Wang, Nan Zhang, Heqing Huang, Wei Zou, and Peng Liu. 2015. Finding unknown malice in 10 seconds: Mass vetting for new threats at the Google-Play scale. In *Usenix Security 15*. 659–674.

[14] Xin Chen and Sencun Zhu. 2015. DroidJust: Automated functionality-aware privacy leakage analysis for Android applications. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. ACM, 5.

[15] Zhenxiang Chen, Qiben Yan, Hongbo Han, Shanshan Wang, Lizhi Peng, Lin Wang, and Bo Yang. 2018. Machine learning based mobile malware detection using highly imbalanced network traffic. *Information Sciences* 433 (2018), 346–364.

[16] Mauro Conti, Qian Qian Li, Alberto Maragno, and Riccardo Spolaor. 2018. The dark side (-channel) of mobile devices: A survey on network traffic analysis. *IEEE Communications Surveys & Tutorials* 20, 4 (2018), 2658–2713.

[17] Shuaifu Dai, Alok Tongaonkar, Xiaoyin Wang, Antonio Nucci, and Dawn Song. 2013. NetworkProfiler: Towards automatic fingerprinting of Android apps. *Proceedings - IEEE INFOCOM*, 809–817.

[18] William Denniss and John Bradley. 2016. OAuth 2.0 for native apps. *Internet Engineering Task Force, Internet-Draft draft-ietf-oauthnative-apps-05* (2016).

[19] Google Developers. 2019. Mixed content weakens HTTPS. https://developers.google.com/web/fundamentals/security/prevent-mixed-content/what-is-mixed-content.

[20] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. 2014. TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)* 32, 2 (2014), 5.

[21] James Eyers. 2017. Cyber Security Minister says firms need to tell customers more about threats. https://www.afr.com/technology/cyber-security-minister-says-firms-need-to-tell-customers-more-about-threats-20170422-gvqbl7.

[22] Farsight Security, Inc. 2019. DNS Database. https://www.dnsdb.info/.

[23] Adrienne Porter Felt, Matthew Finifter, Erika Chin, Steve Hanna, and David Wagner. 2011. A survey of mobile malware in the wild. In *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*. 3–14.

[24] Aditya Grover and Jure Leskovec. 2016. Node2Vec: Scalable feature learning for networks. In *KDD'16* (San Francisco, California, USA). New York, NY, USA, 855–864.

[25] Thorsten Holz, Christian Gorecki, Konrad Rieck, and Felix C. Freiling. 2008. Measuring and detecting fast-flux service networks. In *NDSS*.

[26] Boyang Hu, Qicheng Lin, Yao Zheng, Qiben Yan, Matthew Troglia, and Qingyang Wang. 2019. Characterizing location-based mobile tracking in mobile ad networks. *arXiv preprint arXiv:1903.09916* (2019).

[27] Muhammad Ikram, Narseo Vallina-Rodriguez, Suranga Seneviratne, Mohamed Ali Kaafar, and Vern Paxson. 2016. An analysis of the privacy and security risks of Android VPN permission-enabled apps. In *IMC*. ACM, 349–364.

[28] Paul Jaccard. 1912. The distribution of the flora in the alpine zone. 1. *New Phytologist* 11, 2 (1912), 37–50.

[29] Ruofan Jin and Bing Wang. 2013. Malware detection for mobile devices using software-defined networking. In *2013 Second GENI Research and Educational Experiment Workshop*. IEEE, 81–88.

[30] Issa M. Khalil, Bei Guan, Mohamed Nabeel, and Ting Yu. 2018. A domain is only as good as its buddies: Detecting stealthy malicious domains via graph inference. In *CODASPY*. ACM, 330–341.

[31] Platon Kotzias, Juan Caballero, and Leyla Bilge. 2021. How did that get in my phone? Unwanted app distribution on Android devices. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 53–69.

[32] Bum Jun Kwon, Jayanta Mondal, Jiyong Jang, Leyla Bilge, and Tudor Dumitraş. 2015. The dropper effect: Insights into malware distribution with downloader graph analytics. In *CCS*. ACM, 1118–1129.
[33] Charles Lever, Manos Antonakakis, Bradley Reaves, Patrick Traynor, and Wenke Lee. 2013. The core of the matter: Analyzing malicious traffic in cellular carriers. In *NDSS*.
[34] Jyoti Malik and Rishabh Kaushal. 2016. CREDROID: Android malware detection by network traffic analysis. In *Proceedings of the 1st ACM Workshop on Privacy-Aware Mobile Computing*. ACM, 28–36.
[35] Pratyusa K. Manadhata, Sandeep Yadav, Prasad Rao, and William Horne. 2014. Detecting malicious domains via graph inference. In *European Symposium on Research in Computer Security*. Springer, 1–18.
[36] Claudio Marforio, Ramya Jayaram Masti, Claudio Soriente, Kari Kostiainen, and Srdjan Capkun. 2015. Personalized security indicators to detect application phishing attacks in mobile platforms. *arXiv preprint arXiv:1502.06824* (2015).
[37] McAfee. 2019. McAfee mobile threat report 2019. (2019).
[38] Yisroel Mirsky, Asaf Shabtai, Lior Rokach, Bracha Shapira, and Yuval Elovici. 2016. Sherlock vs Moriarty: A smartphone dataset for cybersecurity research. In *Proc. of the 2016 ACM Workshop on Artificial Intelligence and Security*. 1–12.
[39] Stanislav Miskovic, Gene Moo Lee, Yong Liao, and Mario Baldi. 2015. AppPrint: Automatic fingerprinting of mobile applications in network traffic. In *International Conference on Passive and Active Network Measurement*. Springer, 57–69.
[40] Pejman Najafi, Alexander Mühle, Wenzel Pünter, Feng Cheng, and Christoph Meinel. 2019. MalRank: A measure of maliciousness in SIEM-based knowledge graphs. In *ACSAC*. 417–429.
[41] Fairuz Amalina Narudin, Ali Feizollah, Nor Badrul Anuar, and Abdullah Gani. 2016. Evaluation of machine learning classifiers for mobile malware detection. *Soft Computing* 20, 1 (2016), 343–357.
[42] Alina Oprea, Zhou Li, Ting-Fang Yen, Sang H. Chin, and Sumayah Alrwais. 2015. Detection of early-stage enterprise infection by mining large-scale log data. In *DSN*. IEEE, 45–56.
[43] Elias P. Papadopoulos, Michalis Diamantaris, Panagiotis Papadopoulos, Thanasis Petsas, Sotiris Ioannidis, and Evangelos P. Markatos. 2017. The long-standing privacy debate: Mobile websites vs mobile apps. In *WWW*. 153–162.
[44] Roberto Perdisci, Wenke Lee, and Nick Feamster. 2010. Behavioral clustering of HTTP-based malware and signature generation using malicious network traces. In *NSDI*, Vol. 10. 14.
[45] Andrea Possemato and Yanick Fratantonio. 2020. Towards HTTPS everywhere on Android: We are not there yet. In *USENIX Security'20*. USENIX Association, 343–360.
[46] Babak Rahbarinia, Marco Balduzzi, and Roberto Perdisci. 2016. Real-time detection of malware downloads via large-scale URL file machine graph mining. In *ASIACCS*. ACM, 783–794.
[47] Gyan Ranjan. 2015. SAMPLES: Self adaptive mining of persistent lexical snippets for classifying mobile application traffic.
[48] Gyan Ranjan, Alok Tongaonkar, and Ruben Torres. 2016. Approximate matching of persistent lexicon using search-engines for classifying mobile app traffic. In *IEEE INFOCOM*. IEEE, 1–9.
[49] Jingjing Ren, Martina Lindorfer, Daniel J. Dubois, Ashwin Rao, David Choffnes, and Narseo Vallina-Rodriguez. 2018. Bug fixes, improvements, ... and privacy leaks. (2018).
[50] Jingjing Ren, Ashwin Rao, Martina Lindorfer, Arnaud Legout, and David Choffnes. 2016. ReCon: Revealing and controlling PII leaks in mobile network traffic. In *14th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 361–374.
[51] Kevin A. Roundy, Paula Barmaimon Mendelberg, Nicola Dell, Damon McCoy, Daniel Nissani, Thomas Ristenpart, and Acar Tamersoy. 2020. The many kinds of creepware used for interpersonal attacks. In *IEEE S&P*.
[52] Asaf Shabtai, Lena Tenenboim-Chekina, Dudu Mimran, Lior Rokach, Bracha Shapira, and Yuval Elovici. 2014. Mobile malware detection through analysis of deviations in application network behavior. *Computers & Security* 43 (2014), 1–18.
[53] Mahmood Sharif, Jumpei Urakawa, Nicolas Christin, Ayumu Kubota, and Akira Yamada. 2018. Predicting impending exposure to malicious content from user behavior. In *CCS*. ACM, 1487–1501.
[54] Gianluca Stringhini, Yun Shen, Yufei Han, and Xiangliang Zhang. 2017. Marmite: Spreading malicious file reputation through download graphs. In *ACSAC*. ACM, 91–102.
[55] Acar Tamersoy, Kevin Roundy, and Duen Horng Chau. 2014. Guilt by association: Large scale malware detection by mining file-relation graphs. In *KDD*. ACM, 1524–1533.
[56] Jinjun Tang, Yinhai Wang, Hua Wang, Shen Zhang, and Fang Liu. 2014. Dynamic analysis of traffic time series at different temporal scales: A complex networks approach. *Physica A: Statistical Mechanics and Its Applications* 405 (2014), 303–315.
[57] Vincent F. Taylor, Riccardo Spolaor, Mauro Conti, and Ivan Martinovic. 2016. AppScanner: Automatic fingerprinting of smartphone apps from encrypted network traffic. In *Euro S&P*. IEEE, 439–454.

[58] Vincent F. Taylor, Riccardo Spolaor, Mauro Conti, and Ivan Martinovic. 2017. Robust smartphone app identification via encrypted network traffic analysis. *IEEE Transactions on Information Forensics and Security* (2017).
[59] Alok Tongaonkar, Shuaifu Dai, Antonio Nucci, and Dawn Song. 2013. Understanding mobile app usage patterns using in-app advertisements. In *International Conference on Passive and Active Network Measurement*. Springer, 63–72.
[60] Patrick Traynor, Michael Lin, Machigar Ongtang, Vikhyath Rao, Trent Jaeger, Patrick McDaniel, and Thomas La Porta. 2009. On cellular botnets: Measuring the impact of malicious devices on a cellular network core. In *CCS*. ACM, 223–234.
[61] Thijs van Ede, Riccardo Bortolameotti, Andrea Continella, Jingjing Ren, Daniel J. Dubois, Martina Lindorfer, David Choffnes, Maarten van Steen, and Andreas Peter. [n. d.]. FLOWPRINT: Semi-supervised mobile-app fingerprinting on encrypted network traffic. ([n. d.]).
[62] Eline Vanrykel, Gunes Acar, Michael Herrmann, and Claudia Diaz. 2017. Leaky birds: Exploiting mobile application traffic for surveillance. 367–384.
[63] Verizon. 2019. Mobile Security Index. (2019).
[64] VirusTotal. 2019. VirusTotal. http://www.virustotal.com.
[65] Thomas Vissers, Jan Spooren, Pieter Agten, Dirk Jumpertz, Peter Janssen, Marc Van Wesemael, Frank Piessens, Wouter Joosen, and Lieven Desmet. 2017. Exploring the ecosystem of malicious domain registrations in the .eu TLD. In *Research in Attacks, Intrusions, and Defenses*. Springer International Publishing, 472–493.
[66] Haoyu Wang, Zhe Liu, Jingyue Liang, Narseo Vallina-Rodriguez, Yao Guo, Li Li, Juan Tapiador, Jingcun Cao, and Guoai Xu. 2018. Beyond Google Play: A large-scale comparative study of Chinese Android app markets. In *IMC 2018* (Boston, MA, USA). ACM, 293–307.
[67] Shanshan Wang, Zhenxiang Chen, Lei Zhang, Qiben Yan, Bo Yang, Lizhi Peng, and Zhongtian Jia. 2016. TrafficAV: An effective and explainable detection of mobile malware behavior using network traffic. In *IwQoS*. IEEE, 1–6.
[68] Fengguo Wei, Yuping Li, Sankardas Roy, Xinming Ou, and Wu Zhou. 2017. Deep ground truth analysis of current Android malware. In *International Conf. on Detection of Intrusions and Malware, and Vulnerability Assessment*. 252–276.
[69] Ning Xia, Han Hee Song, Yong Liao, Marios Iliofotou, Antonio Nucci, Zhi-Li Zhang, and Aleksandar Kuzmanovic. 2013. Mosaic: Quantifying privacy leakage in mobile networks. In *ACM SIGCOMM Computer Communication Review*, Vol. 43. 279–290.
[70] Lok-Kwong Yan and Heng Yin. 2012. DroidScope: Seamlessly reconstructing the OS and Dalvik semantic views for dynamic Android malware analysis. In *USENIX Security Symposium*. 569–584.
[71] Chao Yang, Zhaoyan Xu, Guofei Gu, Vinod Yegneswaran, and Phillip Porras. 2014. DroidMiner: Automated mining and characterization of fine-grained malicious behaviors in Android applications. In *European Symposium on Research in Computer Security*. Springer, 163–182.
[72] Jaemin Yoo, Saehan Jo, and U Kang. 2017. Supervised belief propagation: Scalable supervised inference on attributed networks. In *ICDM*. IEEE, 595–604.
[73] Apostolis Zarras, Antonis Papadogiannakis, Robert Gawlik, and Thorsten Holz. 2014. Automated generation of models for fast and precise detection of HTTP-based malware. In *PST*. IEEE, 249–256.
[74] Min Zhao, Tao Zhang, Fangbin Ge, and Zhijian Yuan. 2012. RobotDroid: A lightweight malware detection framework on smartphones. *Journal of Networks* 7, 4 (2012), 715.
[75] Xiaojin Zhu and Zoubin Ghahramani. 2002. Learning from labeled and unlabeled data with label propagation.
[76] Zhichao Zhu, Guohong Cao, Sencun Zhu, Supranamaya Ranjan, and Antonio Nucci. 2012. A social network based patching scheme for worm containment in cellular networks. In *Handbook of Optimization in Complex Networks*. Springer, 505–533.

Received 19 January 2021; revised 21 June 2022; accepted 17 August 2022