Leaky Birds: Exploiting Mobile Application Traffic for Surveillance

Eline Vanrykel[1], Gunes Acar[2], Michael Herrmann[2], and Claudia Diaz[2]

[1] KU Leuven, Leuven, Belgium
eline.vanrykel@gmail.com
[2] KU Leuven ESAT/COSIC, iMinds, Leuven, Belgium
{name.surname}@esat.kuleuven.be

Abstract. Over the last decade, mobile devices and mobile applications have become pervasive in their usage. Although many privacy risks associated with mobile applications have been investigated, prior work mainly focuses on the collection of user information by application developers and advertisers. Inspired by the Snowden revelations, we study the ways mobile applications enable mass surveillance by sending unique identifiers over unencrypted connections. Applying passive network fingerprinting, we show how a passive network adversary can improve his ability to target mobile users' traffic.
Our results are based on a large-scale automated study of mobile application network traffic. The framework we developed for this study downloads and runs mobile applications, captures their network traffic and automatically detects identifiers that are sent in the clear. Our findings show that a global adversary can link 57% of a user's unencrypted mobile traffic. Evaluating two countermeasures available to privacy aware mobile users, we find their effectiveness to be very limited against identifier leakage.

1 Introduction

Documents that have been revealed by the former NSA contractor Edward Snowden shed light on the massive surveillance capabilities of the USA and UK intelligence agencies. One particular document released by the German newspaper *Der Spiegel* describes the ways in which traffic of mobile applications (apps) is exploited for surveillance [16]. The document, which reads "Exploring and Exploiting Leaky Mobile Apps With BADASS," provides a unique opportunity to understand the capabilities of powerful network adversaries. Furthermore, the document reveals that identifiers sent over unencrypted channels are being used to distinguish the traffic of individual mobile users with the help of so-called *selectors*. Similar revelations about the use of Google cookies to target individuals imply that BADASS is not an isolated incident [12, 34].

While it is known that a substantial amount of mobile app traffic is unencrypted and contains sensitive information such as users' location or real identities [24,35,43], the opportunities that mobile traffic offers to surveillance agencies

may still be greatly underestimated. Identifiers that are being sent in the clear, may allow the adversary to link app sessions of users and thus to learn more information about the surveilled users than he could without. The purpose of this study is to evaluate this risk and to quantify the extent to that it is possible to track mobile app users based on unencrypted app traffic.

To this end we present a novel framework to quantify the threat that a surveillance adversary poses to smartphone users. The framework automates the collection and analysis of mobile app traffic: it downloads and installs Android apps, runs them using Android's *The Monkey* [18] tool, captures the network traffic on cloud-based VPN servers, and finally analyzes the traffic to detect unique and persistent identifiers. Our framework allows large-scale evaluation of mobile apps in an automated fashion, which is demonstrated by the evaluation of 1260 apps. We choose the apps among all possible categories of the Google Play store and of different popularity levels.

Our study is inspired by a recent work by Englehardt *et al.* [26]. They studied the surveillance implications of cookie-based tracking by combining web and network measurements. The evaluation method they use boils down to measuring the success of the adversary by the ratio of user traffic he can cluster together. We take a similar approach for automated identifier detection but we extend their work to capture non-cookie-based tracking methods that are suitable for user tracking. Moreover, we show how TCP timestamp-based passive network fingerprinting can be used to improve the clustering of the traffic and may allow to detect the boot time of Android devices.

1.1 Contributions

Large-scale, automated study on surveillance implications of mobile apps. We present an automated analysis of 1260 Android apps from 42 app categories and show how mobile apps enable third party surveillance by sending unique identifiers over unencrypted connections.

Applying passive network fingerprinting for mobile app traffic exploitation. We show how a passive network adversary can use TCP timestamps to significantly improve the amount of traffic he can cluster. This allows us to present a more realistic assessment of the threat imposed by a passive adversary. Further, we show how an adversary can guess the boot time of an Android device and link users' traffic even if they switch from WiFi to 3G or vice versa.

Evaluation of the available defenses for privacy aware users. We analyze the efficacy of two mobile ad-blocking tools: Adblock Plus for Android [13] and Disconnect Malvertising [14]. Our analysis shows that these tools have a limited effect preventing mobile apps from leaking identifiers.

2 Background and Related Work

Table 1: Unique smartphone identifiers present on Android, an overview.

Name	**Persistence**	**Permission**
Android ID	until a factory reset	None
MAC Address	lifetime of the device	`ACCESS_WIFI_STATE`
IMEI	lifetime of the device	`READ_PHONE_STATE`
IMSI	lifetime of the SIM card	`READ_PHONE_STATE`
Serial number	lifetime of the device	None [41]
SIM serial number	lifetime of the SIM card	`READ_PHONE_STATE`
Phone number	lifetime of the SIM card	`READ_PHONE_STATE`
Google Advertising ID	until reset by the user	`ACCESS_NETWORK_STATE`, `INTERNET`

Android apps and identifiers. Android apps and third-parties can access common identifiers present on the smartphone, such as MAC address, Google Advertising ID or IMEI number. We call these identifiers *smartphone IDs*. An overview of the Android smartphone IDs can be found in Table 1. Developers may also choose to assign IDs to users (instead of using smartphone IDs), for identifying individual app installations or simply to avoid asking for additional permissions [11]. We refer to such identifiers as *app assigned IDs*.

Privacy implications of mobile apps. Although privacy implications of Android apps have been extensively studied in the literature [25, 28, 29], prior work has mainly focused on the sensitive information that is collected and transmitted to remote servers. Xia et al. showed that up to 50% of the traffic can be attributed to the real names of users [43]. Enck et al. developed TaintDroid [25], a system-wide taint analysis system that allows runtime analysis and tracking of sensitive information flows. While it would be possible to use TaintDroid in our study, we opted to keep the phone modifications minimal and collect data at external VPN servers. This allows us to have a more realistic assessment of application behavior and adversary capabilities.

Our work differs from these studies, by quantifying the threat posed by a passive network adversary who exploits mobile app traffic for surveillance purposes. We also show how the adversary can automatically discover user identifiers and use passive network fingerprinting techniques to improve his attack.

Passive network monitoring and surveillance. Englehardt et al. [26] show how third-party cookies sent over unencrypted connections can be used to *cluster* the traffic of individual users for surveillance. They found that reconstructing 62-73% of the user browsing history is possible by passively observing network traffic.

In addition to using identifiers to track smartphones, an eavesdropping adversary can use passive network fingerprinting techniques to distinguish traffic from different physical devices. Prior work showed that clock skew [31, 33, 44], TCP timestamps [23, 42] and IP ID fields [21] can be used to remotely identify hosts or count hosts behind a NAT. In this study, we use TCP timestamps to

improve the linking of users' mobile traffic in short time intervals. We assume the adversary to exploit TCP timestamps to distinguish traffic of users who are behind a NAT. Moreover, we demonstrate how an adversary can discover the boot time of an Android device by exploiting TCP timestamps.

3 Threat Model

In this paper we consider passive network adversaries whose goal is to link app traffic of smartphone users. The adversaries observe unique identifiers that are being transmitted from mobile apps in the clear and apply network fingerprinting techniques. We consider that the adversaries cannot break cryptography or launch MITM attacks such as SSLstrip [32].

We distinguish between two types of passive adversaries: A *global passive adversary*, who can intercept all Internet traffic at all time; and a *restricted passive adversary* who can only observe a limited part of the network traffic. Both adversaries have the capability to collect bulk data. This may be achieved in various ways, such as tapping into undersea fiber-optic cables; hacking routers or switches; intercepting traffic at major Internet Service Providers (ISP) or Internet Exchange Points (IXP) [3].

There can be several models in which an adversary may have limited access to the user's traffic. In this study we evaluate adversaries whose limitation is either *host-based* or *packet-based*. The host-based adversary is only able to see traffic bound to certain hosts; for example, because the adversary is only able to obtain warrants for intercepting traffic within its own jurisdiction. The packet-based adversary may only have access to a certain point in the Internet backbone and thus miss traffic that is being sent along other routes. For both adversaries, we evaluate the success based on different levels of network coverage (Section 7.2). We simulate partial network coverage by randomly selecting hosts or packets to be analyzed depending on the model. For instance, for the host-based model with 0.25 network coverage, we randomly pick one-fourth of the hosts and exclude the traffic bound to remaining hosts from the analysis.

4 Data Collection Methodology

4.1 Experimental Setup

We present the experimental setup[4] that is used for this paper in Fig. 1. It includes a controller PC, two smartphones and two VPN servers for traffic capture. The main building blocks of our framework are as follows:

[3] All these methods are feasible, as illustrated by the Snowden revelations [6, 8].

[4] The source code of the framework, as well as the collected data will be made available to researchers upon request.

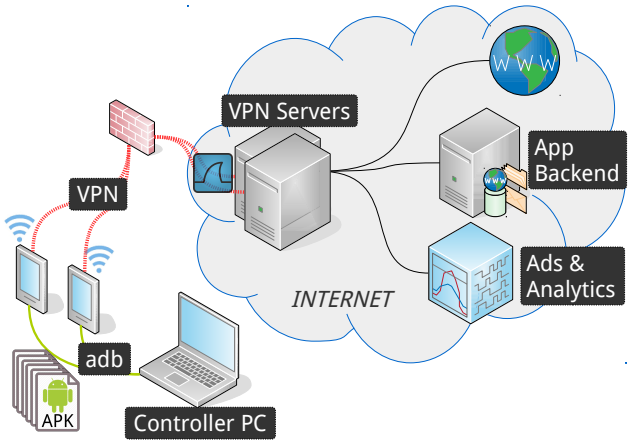


Fig. 1: Our setup in this study consists of a Controller PC that manages the experiments, two Android phones that run apps, and two VPN servers that capture the network traffic.

Controller PC. The Controller PC runs the software that orchestrates the experiments and the analysis. It has three main tasks: 1) installing apps on the smartphones and ensuring that the experiment runs smoothly, e.g. checking the phone's WiFi and VPN connections, 2) sending SSH commands to the remote VPN servers to start, stop and download the traffic capture, 3) analyzing the collected data.

Smartphones. We conducted our experiments with two *Samsung Galaxy SIII Mini* smartphones running Android version 4.1.2. We *rooted* the phones to address issues such as storage and uninstallation problems. Although we considered using the Android emulator as in other works [24, 36, 38], our preliminary tests [39] showed that the number of transmitted identifiers is significantly less in the emulator compared to the same setting with a real smartphone and the emulator lacks certain identifiers, such as the WiFi MAC address. We also chose not to intercept system API calls or instrument the operating system, such as in [25, 27], since we preferred a simpler and more portable solution.

The Monkey. We use *The Monkey* [18] tool to automate the experiments and simulate the user interaction at large scale. The Monkey generates a pseudo-random event stream that includes touch, motion and keyboard events.

Traffic Capture. The network traffic is captured by two remote VPN servers, using the *dumpcap* [5] command line tool. Using VPN servers, we could capture *all* the network traffic and not only HTTP traffic, which would be the case with an HTTP proxy. Also, since we record the traffic on remote machines, we ensure that there is no packet drop due to lack of buffer space on resource constrained devices [15]. However, we captured traffic locally on the phone during the evaluation of ad-blockers for Android. These tools use a proxy or VPN

themselves to block ads. Since Android does not allow simultaneous VPN connections, we captured the traffic locally by running *tcpdump* on the smartphones. To ensure comparability, we exclude all the captures where we observed packet drops from the analysis (20% of the cases, 171 apps in two experiments).

Traffic parser. For parsing the captured network traffic, we developed a Python script based on the `dpkt` [3] packet parsing library. The script allows us to decode IPv4 and IPv6 datagrams, reassemble TCP streams, decompress compressed HTTP bodies and to parse GRE and PPTP encapsulation used by the VPN. We extract HTTP headers and bodies, packet timestamps, IP addresses and port numbers from the packets for later use. Since it is outside of the scope of this study, we did not decrypt SSL/TLS records. However, for the TCP timestamp analysis described in Section 6 it is beneficial, yet not necessary, to extract TCP timestamps from all TCP packets, including the ones from encrypted HTTPS traffic. Note that this is within our adversary model, because TCP headers are sent in the clear and thus available to a passive adversary.

Having described the main building blocks of the experimental setup, now we outline the different modes and steps of the experiments:

Experiment modes. We run experiments in two different modes to evaluate the difference in identifier transmission; i) if the app is simply opened and ii) if the user actually interacts with the app. We refer to the former as *startscreen experiment* and to the latter as *interactive experiment*. The Monkey is used to simulate user interaction in the interactive experiments.

Evaluation of ad-blocker apps. We evaluate the effect of apps that block ads and trackers. While those apps are not specifically designed to prevent identifier leakage, they may still reduce the number of identifiers being sent in the clear. Specifically, we repeated the experiment of the top-popularity apps after we installed and activated the ad-blocker apps *Adblock Plus for Android* [13] and *Disconnect Malvertising* [14].

Steps of the experiment. Our framework executes the steps of the experiments in an entirely automated fashion. The Controller PC connects the smartphone to the VPN server by running a Python based *AndroidViewClient* [4] script that emulates the touch events necessary to start the VPN connection on the smartphone. Since installing all the apps at once is not possible due to storage constraints, our framework conducts the experiment in cycles. In each cycle we install 20 apps and then run them sequentially[5]. The apps for each cycle are randomly chosen from the entire set of apps, with the condition that each app is only picked once. Before running an app, the Controller PC kills the process of the previous app. This way we are able to prevent the traffic of the previously tested app mistakenly being recorded for the subsequent app. After finished running the 20 apps, the Controller PC runs all 20 apps a second time in the same order. Running each app twice enables the automated detection of identifiers outlined in Section 5.1. Finally, the Controller PC completes the current cycle by uninstalling all 20 apps.

[5] We chose 20 since this was the maximum number of apps that can be installed on an Android emulator at once, which we used in the preliminary stages of the study.

4.2 Obtaining Android Applications

To obtain the Android apps, we developed scripts for crawling the Google Play store and, subsequently, to download APK files. Our scripts are based on the Python Selenium [17] library, the `APK downloader` browser extension and webpages [1]. Using this software, we crawled the entire Play Store and obtained information on $1,003,701$ different Android apps. For every app we collected information such as number of downloads, rating scores and app category. This allows us to rank the apps of every category according to their popularity.

For every app category we choose 10 apps from three different popularity levels: *top-popularity*, *mid-popularity* and *low-popularity*. While we use the most popular apps for the top-popularity category, we sample the mid-popularity and low-popularity apps from the 25th and 50th percentiles from each category. At the time we conducted the crawl, there were 42 different app categories and we therefore obtained a total of 1260 ($42 \times 10 \times 3$) apps. The average time for evaluating one app is 64 seconds.

5 Analysis Methodology

In the following we show how an adversary is able to extract identifiers from network traffic and then use these identifiers to cluster data streams, i.e. linking data streams as belonging to the same user. This is the same that an adversary with the goal of surveilling Internet traffic would do, i.e. extracting and applying a set of *selectors* that match unique and persistent mobile app identifiers.

5.1 Identifier Detection

Suitable identifiers for tracking need to be persistent and unique, i.e. the same ID cannot appear on different phones and IDs need to be observable over multiple sessions. Our framework automatically detects such unique identifiers in unencrypted mobile app traffic. While the overall approach is similar to the one in [19,26] we extend the cookie-based identifier detection technique to cover mobile app traffic. We assume that the smartphone IDs (such as Android ID or MAC address) are not known a priori to the adversary. The adversary has to extract IDs based on the traffic traces only. Yet, we use smartphone IDs as the ground truth to improve our automated ID detection method by tuning its parameters.

For finding identifiers, we process HTTP request headers, bodies and URL parameters. Specifically, the steps of the unique identifier detection are as follows:

- Split URLs, headers, cookie contents and message bodies using common delimiters, such as “=”, “&”, “:”, to extract key-value pairs. Decode JSON encoded strings in HTTP message bodies.

- Filter out cookies with expiry times shorter than three months. A tracking cookie is expected to have a longer expiry period [26].
- For each key-value pair, we construct an *identifying rule set* and add it to the potential identifier list. This is the tuple (host, position, key), where *host* is extracted from the HTTP message and *position* indicates whether the key was extracted from a cookie, header or URL.
- Compare values of the same key between runs of two smartphones.
  - Eliminate values if they are not the same length.
  - Eliminate values that are not observed in two runs of the same app on the same smartphone.
  - Eliminate values that are shorter than 10 or longer than 100 characters.
  - Eliminate values that are more than 70% similar according to the Ratcliff-Obershelp similarity measure [22].
- Add (host, position, key) to the rule set.

We tuned similarity (70%) and length limits (10, 100) according to two criteria: minimizing false positives and detecting all the smartphone identifiers (Table 1) with the extracted rule set. We experimented with different limit values and picked the values that gave us the best results based on these criteria. A more thorough evaluation of these limits is omitted due to space constraints, but interested readers can refer to [19,26] for the main principles of the methodology.

5.2 Clustering of App Traffic

While the ultimate goal of the adversary is to link different app sessions of the same user by exploiting unique identifiers transmitted in app traffic, the first challenge of the adversary is to identify the traffic of one app. An app may open multiple TCP connections to different servers and linking these connections can be challenging. The user's public IP address is not a good identifier: several users may share the same public IP via a NAT. Moreover, IP addresses of mobile phones are known to change frequently [20].

In this work we consider two different clustering strategies. In the *TCP stream based linking*, the attacker can only link IP packets based on their TCP stream. The adversary can simply monitor creation and tear down of TCP streams and ensure that the packets being sent within one stream are originating from the same phone. The second, more sophisticated strategy employs passive network fingerprinting techniques to link IP packets of the same app session. We will refer this technique as *app session based linking* and outline it in Section 6.

Following Englehardt et al. [26] we present linking of the user traffic as a graph building process. We use the term *node* to refer to a list of packets that the adversary is certain that they belong to the same user. As explained above, this is either a TCP stream or an app session. For every node the adversary extracts the identifying rule set (host, position, key) as described in Section 5.1. Starting from these nodes, the adversary inspects the content of the traffic and then tries to link nodes together to so-called *components*.

Therefore, the attacker will try to match a node's identifiers to the identifiers of the existing components. We account for the fact that some developers do not use the smartphone ID right away as identifier, but derive an identifier from it by hashing or encoding. Thus the clustering algorithm will also try to match the SHA-1, SHA-256, MD5 and murmur3 hashes and base64 encoded form of the identifiers. For every node, there exist three possibilities when comparing the node's identifiers to a existing component's identifiers:

1. **The node's value (or its derivative) matches the identifiers of an existing component:** The node will be added to the component and the respective identifiers are being merged, i.e. the newly added node may include identifiers not yet included in the component.
2. **None of the node's identifiers or their derivatives can be matched to an existing component:** The node creates its own component which is disconnected from all other components.
3. **The node shares identifiers with multiple components:** These components are merged together and the node is added to this component.

For the remainder of this paper, we refer to the component that contains the highest number of nodes as the *Giant Connected Component* (GCC). Furthermore, we define the ratio of number of nodes in GCC to the number of nodes in the whole graph as the *GCC ratio*. The *GCC ratio* serves as a metric for measuring the adversary's success for linking users' traffic based on the amount of traffic he observes.

5.3 Background Traffic Detection

The Android operating system itself also generates network traffic, for example to check updates or sync user accounts. Although we find in our experiments that the Android OS does not send any identifiers in the clear, we still take measures to avoid that this traffic pollutes our experiment data. Particularly, we captured the network traffic of two smartphones for several hours multiple times without running any app. A complete overview of all HTTP queries made during such captures can be found in [40]. We excluded all the HTTP requests to these domains during the analysis stage. Although we excluded background traffic from our analysis, the adversary may try to exploit the background traffic in a real-world attack.

6 Linking Mobile App Traffic with TCP Timestamps

In this section we elaborate on the adversary's ability to employ passive fingerprinting techniques to link different IP packets originating from the same smartphone. As mentioned in Section 5.2, this gives a significant advantage to the adversary when clustering the user traffic. In particular, the adversary is

able to analyze TCP timestamps for this task as they are commonly allowed by the firewalls [33].

TCP timestamps are an optional field in TCP packets that include a 32-bit monotonically increasing counter. They are used to improve the protocol performance and protect against old segments that may corrupt TCP connections [30]. While the exact usage of TCP timestamps is platform dependent, our inspection of the Android source code and capture files from our experiments revealed that Android initializes the TCP timestamp to a fixed value after boot and uses 100Hz as the timestamp increment frequency [2]. Thus, at any time t, TCP timestamp of a previously observed device can be estimated as follows: $TS_t = TS_{prev} + 100 \times (t - t_{prev})$, where TS_{prev} is the timestamp observed at t_{prev} and $(t - t_{prev})$ is the elapsed time. The adversary can therefore link different visits from the same device by comparing the observed TCP timestamps to his estimate. Prior studies have shown that distinguishing devices behind a NAT using TCP timestamps can be done in an efficient and scalable manner [23, 37, 42].

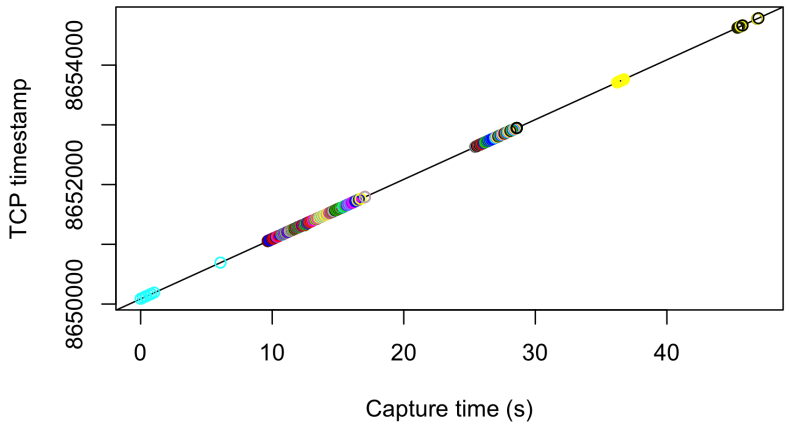


Fig. 2: TCP timestamp vs. capture time plot of Angry Birds Space app follows a line with a slope of 100, which is the timestamp resolution used by Android. Different TCP sessions, indicated by different colors, can be linked together by exploiting the linearity of the TCP timestamp values.

Fig. 2 demonstrates the linear increase of the TCP timestamps of a phone running the "Angry Bird Space" app. To demonstrate the linkability of TCP streams, every point in Fig. 2 is colored based on its TCP source and destination port. The straight line shows that the adversary can easily link different TCP streams of the same device by exploiting the linearity of the timestamps. The

adversary is also able to consider TCP timestamps of encrypted communications, because TCP timestamps are sent unencrypted in the packet headers. This allows adversaries within our threat model to further increase the success of the linking. Furthermore, TCP timestamps can be used to link traffic even if the user switches from WiFi to mobile data connection or vice versa [40]. Finally, the linking is still feasible even if the adversary misses some packets, for instance, due to partial coverage of the network.

Limitations. During the background traffic detection experiments, we observed cases in which TCP timestamps are not incremented linearly. Consulting the Android System Clock Documentation, we determined that the CPU and certain system timers stop when the device enters the *deep sleep* state [10]. This power saving mechanism is triggered only when the screen is off and the device is not connected to the power outlet or USB port. Therefore, the phone will never go into deep sleep when a user is interacting with an app and the TCP timestamps will be incremented in a predictable way, allowing the linking of the traffic by app sessions.

Implications for traffic linking. We will assume the adversary can use TCP timestamps to cluster packets generated during the use of an app (app session), as the phone never enters *deep sleep* mode when it is in active use. As mentioned in Section 5.2, we will refer to this as *app session based linking*.

Android boot time detection. In addition to linking packets from different TCP streams, TCP timestamps can also be used to guess the boot time of remote devices [7]. Among other things, boot time can be used to determine if the device is patched with critical updates that require a reboot. Since it is not directly related to traffic linking attack considered in the study, we explain the boot time detection methodology in the unabridged version of this paper [40].

7 Results

7.1 Identifier Detection Rules

We present in Table 2 an overview of the identifying rule set that we detected by the methodology explained in Section 5.1. Recall that identifying rules correspond to "selectors" in the surveillance jargon, which allow an adversary to target a user's network traffic. In total we found 1597 rules with our method, of which 1127 (71%) correspond to a smartphone ID or its derivative. Our results show that the Android ID and Google Advertising ID are the most frequently transmitted smartphone IDs, accounting for 72% (812/1127) of the total. We group the least commonly transmitted smartphone IDs under the *Other Smartphone IDs* column, which include the following: device serial number, IMSI, SIM serial number and registered Google account email. Furthermore, we found 29% of the extracted rules to be app-assigned IDs.

Table 2: The extracted ID detection rules and corresponding smartphone IDs. *SID*: Smartphone ID, *AAID*: App Assigned ID.

Exp. Mode	**App popularity**	**Android ID**	**Google Ad ID**	**IMEI**	**MAC**	**Other SIDs**	**AAIDs**	**Total ID Rules**
Interactive	top	165	111	63	29	16	193	577
Startscreen	top	115	56	45	19	11	91	337
Interactive	mid	56	28	20	6	5	60	175
Startscreen	mid	48	28	16	5	4	40	141
Interactive	low	73	61	22	15	8	53	232
Startscreen	low	47	24	16	7	8	33	135
Total		504	308	182	81	52	470	1597

Analyzing the extracted rules for the top-popularity, interactive experiments, we found that 50% of the identifiers are sent in the URI of the HTTP requests (291 rules). In 39% (225) of the rules, the IDs are sent in the HTTP request body, using the POST method. Only 3% (18) of the cases, the identifier was sent in a cookie. The average identifier length in our rule set is 26.4 characters. A sample of identifying rules is given in Table 3.

Table 3: Examples rules found in the constructed identifying rule set. The values are modified to prevent the disclosure of real identifiers of the phones used in the study.

Host	**Position**	**Key**	**ID**	**Value**
`data.flurry.com`	Body	offset60	Android ID	AND9f20d23388...
`apps.ad-x.co.uk`	URI	custom_data / meta_udid	Unknown	19E5B4CEE6F5...
`apps.ad-x.co.uk`	URI	macAddress	WiFi MAC	D0:C4:F7:58:6C:12
`alog.umeng.com`	Body	header / device_id	IMEI	354917158514924
`d.applovin.com`	Body	device_info / idfa	Google Ad ID	0e5f5a7d-a3e4-..

After extracting identifier detection rules, we apply them to the traffic captured during the experiments. Due to space constraints we present the detailed results on the transmitted IDs in the unabridged version of this paper [40].

Moreover, analyzing the traffic captures of the top-popularity apps, we found that certain apps send precise location information (29 apps), email address (7 apps) and phone number (2 apps) in the clear. Together with the linking attack presented in this paper, this allows an adversary to link significantly more traffic to real-life identities.

Table 4: The most common third-party hosts found to collect at least an identifier over unencrypted connections. The listed hosts are contacted by the highest number of apps (based on 420 top-popularity apps, interactive experiment).

Host	# Apps	Collected IDs
`data.flurry.com`	43	Android ID
`ads.mopub.com`	32	Google advertising ID
`apps.ad-x.co.uk`	22	Google advertising ID, IMEI, Serial number, Android ID
`alog.umeng.com`	16	IMEI
`a.applovin.com`	16	Google advertising ID

We found that 1076 different hosts were contacted over unencrypted connections during the experiments for the top-popularity apps in the interactive mode. The *data.flurry.com* domain is the most popular third-party domain collecting Android ID from 43 different apps (Table 4). Note that *data.flurry.com* received a notable mention in the slides of the BADASS program [16] for its identifier leakage.

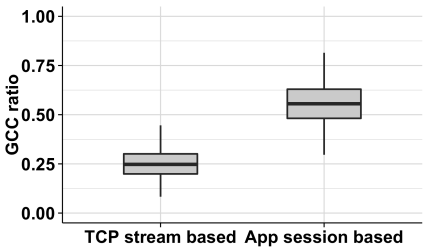
7.2 Traffic Clustering

Here we evaluate the adversary's success in terms of unencrypted app traffic ratio (GCC ratio) that he can link together in different settings. We follow the analysis methodology explained in Section 5.2 and present clustering results for 100 randomly selected combinations of 27 apps. We pick 27 apps since it is the average number of apps used per month according to a recent survey [9]. Running 100 iterations with a different combination of (27) apps allowed us to reduce the variance between different runs and account for all the studied apps. We only consider apps that send at least one HTTP request and calculate the GCC ratio based on the unencrypted traffic. For the top-popular apps in interactive mode, these account for 69% of the apps. For simplicity, we will present the clustering results for only one phone and a single run of each app. The results from two phones did not have any significant difference.

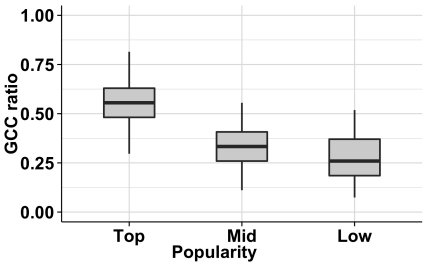
Effect of using TCP timestamps for traffic linking. The left boxplot in Fig. 3(a), shows that when the adversary does not take TCP timestamps into account (TCP stream based linking), he can cluster 25% of users' unencrypted traffic. However, by exploiting TCP timestamps he can increase the GCC ratio to 57%.

Effect of app popularity Fig. 3(b) shows that popularity has a significant impact on the linking success of the adversary. The most popular apps allow the adversary to cluster 57% of the unencrypted traffic, while the apps from the mid-popular and low-popular level result in a GCC ratio of 32% and 28%, respectively.

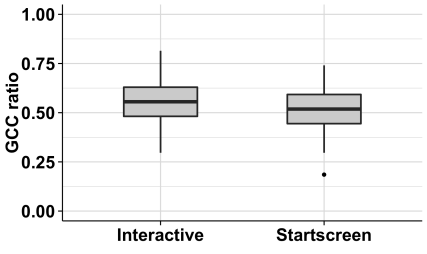
Due to space constraints, we will only present results for the apps from the top-popularity level in the rest of this section.



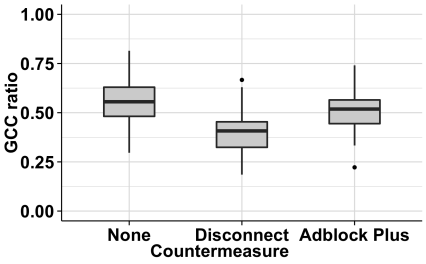
(a) GCC ratio for the top-popularity apps, shown for TCP stream and app session based linking.



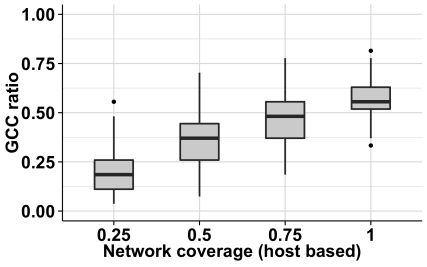
(b) GCC ratio for apps of different popularity levels for interaction mode.



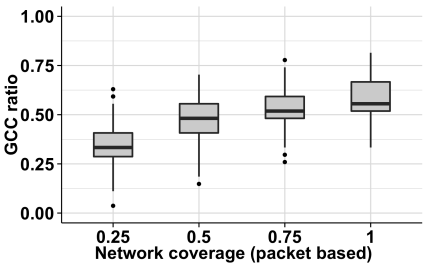
(c) GCC ratio for top-popularity apps, shown for interaction and startscreen mode.



(d) GCC ratio for the top-popularity apps, shown while using different privacy enhancing tools.



(e) GCC ratio for the top-popularity apps, shown for different network coverage levels of a *host based* restricted adversary.



(f) GCC ratio for the top-popularity apps, shown for different network coverage levels of a packet based restricted adversary.

Fig. 3: The success of the adversary under different experimental settings and adversary models. The GCC ratio is the proportion of the unencrypted app traffic that the adversary can link together. The results are shown for 100 different randomly selected combinations of 27 apps.

Effect of user interaction. Fig. 3(c) shows the GCC ratio for two different experiment modes, *interaction* and *startscreen*. Although, the number of identifiers sent in two modes are significantly different (577 vs. 337), the graph shows a similar GCC ratio around 53% for two modes. A possible explanation is that the identifiers that enable linking are already sent as soon as the app is started.

Effect of countermeasures. Fig. 3(d) shows that both ad-blocking apps provide a limited protection against linking of the app traffic. Using Adblock Plus leads to an average linking of 50%. Disconnect Malvertising performs better, with a GCC rate of 38%, reduced from 57%.

Restricted adversary. Fig. 3(e) shows that an adversary that can only intercept traffic to 50% of the hosts can link up to 38% of the unencrypted mobile app sessions. For the *packet based* restricted adversary model, we observe that an adversary with a limited coverage of 25% of the user's packets can still link 37% of all app sessions together (Fig. 3(f)). In both models restricted adversary's success grows almost linear with his network coverage. Note that packet based restricted adversary can link significantly more traffic than the host-based model for the same network coverage ratio. This may be due to being able to observe packets from more hosts which will allow to link apps across sessions.

8 Limitations

Some apps may not be fully discovered by *The Monkey*, leading to an incomplete view of the network traffic. Also, apps that require user logins may not be sufficiently analyzed by our automated methodology. For those reasons, our results should be taken as lower bounds.

While we assume that the smartphones can be distinguished by their TCP timestamps, some middleboxes may interfere with user traffic. Firewalls, proxies or cache servers may terminate outgoing HTTP or TCP connections and open a new connection to the outside servers. Furthermore, end-user NAT devices may have various configurations and hence behave differently compared to enterprise NATs. In such cases, the adversary's ability to link traffic by TCP timestamps may be reduced.

We used *rooted* Android phones in our experiments. Although rooting the phones may introduce changes in the observed traffic, we assumed the changes to be minimal.

9 Conclusion

The revealed slides of the BADASS program have shown that unencrypted mobile app traffic is exploited for mass surveillance. Identifiers sent in the clear by the mobile applications allow targeting mobile users, linking of their traffic and building a database of their online activities.

In this study, we evaluated the surveillance threat posed by a passive network adversary who exploits mobile app traffic for surveillance purposes. We presented a novel framework that automates the analysis of mobile app network traffic. Our framework and methodology is designed to be flexible and can be used in other mobile privacy studies with slight modifications.

Our results show that using TCP timestamps and unique identifiers sent in the unencrypted HTTP traffic, a global adversary can cluster 57% of users' unencrypted mobile app sessions. We demonstrated that a passive adversary can automatically build a rule set that extracts unique identifiers in the observed traffic, which serves as a "selector" list for targeting users.

Our results suggest that popular apps leak significantly more identifiers than the less popular apps. Furthermore, while interacting with the app increases the number of leaked identifiers, solely starting an app amounts to the same attack effectiveness.

We evaluated two countermeasures designed to block mobile ads and found that they provide a limited protection against linking of the user traffic. Encrypting mobile app traffic can effectively protect against passive network adversaries. Moreover, a countermeasure similar to HTTPS Everywhere browser extension can be developed to replace insecure HTTP connections of mobile apps with secure HTTPS connections on the fly.

Acknowledgment

Thanks to Yves Tavernier for sharing his valuable insights about middleboxes, and anonymous reviewers for their helpful and constructive feedback. This work was supported by the Flemish Government FWO G.0360.11N Location Privacy, FWO G.068611N Data mining and by the European Commission through H2020-DS-2014-653497 PANORAMIX and H2020-ICT-2014-644371 WITDOM.

References

1. APK Downloader [Latest] Download Directly — Chrome Extension v3 (Evozi Official). http://apps.evozi.com/apk-downloader/
2. Cross Reference: /external/kernel-headers/original/asm-arm/param.h. http://androidxref.com/4.1.2/xref/external/kernel-headers/original/asm-arm/param.h#18
3. dpkt 1.8.6.2 : Python Package Index. https://pypi.python.org/pypi/dpkt
4. dtmilano/AndroidViewClient. https://github.com/dtmilano/AndroidViewClient/
5. dumpcap - The Wireshark Network Analyzer 1.12.2. https://www.wireshark.org/docs/man-pages/dumpcap.html
6. GCHQ taps fibre-optic cables for secret access to world's communications
7. Nmap Network Scanning - Remote OS Detection - Usage and Examples. http://nmap.org/book/osdetect-usage.html
8. NSA Prism program taps in to user data of Apple, Google and others. http://www.theguardian.com/world/2013/jun/06/us-tech-giants-nsa-data

9. Smartphones: So many apps, so much time. http://www.nielsen.com/us/en/insights/news/2014/smartphones-so-many-apps--so-much-time.html
10. SystemClock — Android Developers. http://developer.android.com/reference/android/os/SystemClock.html
11. Identifying App Installations — Android Developers Blog. http://android-developers.blogspot.be/2011/03/identifying-app-installations.html (2011)
12. 'Tor Stinks' presentation. http://www.theguardian.com/world/interactive/2013/oct/04/tor-stinks-nsa-presentation-document (2013)
13. About Adblock Plus for Android. https://adblockplus.org/android-about (2015)
14. Disconnect Malvertising for Android. https://disconnect.me/mobile/disconnect-malvertising/sideload (2015)
15. Manpage of TCPDUMP. http://www.tcpdump.org/tcpdump_man.html (2015)
16. Mobile apps doubleheader: BADASS Angry Birds. http://www.spiegel.de/media/media-35670.pdf (2015)
17. Selenium - Web Browser Automation. http://docs.seleniumhq.org/ (2015)
18. UI/Application Exerciser Monkey — Android Developers. http://developer.android.com/tools/help/monkey.html (2015)
19. Acar, G., Eubank, C., Englehardt, S.: The Web Never Forgets: Persistent Tracking Mechanisms in the Wild. Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (2014)
20. Balakrishnan, M.: Where's That Phone?: Geolocating IP Addresses on 3G Networks. Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference pp. 294–300 (2009)
21. Bellovin, S.M.: A Technique for Counting NATted Hosts. Proceedings of the second ACM SIGCOMM Workshop on Internet measurment - IMW '02 p. 267 (2002)
22. Black, P.E.: Ratcliff/Obershelp pattern recognition. http://xlinux.nist.gov/dads/HTML/ratcliffObershelp.html (December 2004)
23. Bursztein, E.: Time has something to tell us about network address translation. In: Proc. of NordSec (2007)
24. Dai, S., Tongaonkar, A., Wang, X., Nucci, A., Song, D.: NetworkProfiler: Towards automatic fingerprinting of Android apps. 2013 Proceedings IEEE INFOCOM pp. 809–817 (Apr 2013)
25. Enck, W., Cox, L.P., Gilbert, P., Mcdaniel, P.: TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. OSDI'10 Proceedings of the 9th USENIX conference on Operating systems design and implementation (2010)
26. Englehardt, S., Reisman, D., Eubank, C., Zimmerman, P., Mayer, J., Narayanan, A., Felten, E.W.: Cookies That Give You Away: The Surveillance Implications of Web Tracking. In: Proceedings of the 24th International Conference on World Wide Web. pp. 289–299 (2015)
27. Felt, A.P., Chin, E., Hanna, S., Song, D., Wagner, D.: Android Permissions Demystified. Proceedings of the 18th ACM Conference on Computer and Communications Security p. 627 (2011)
28. Grace, M., Zhou, W., Jiang, X., Sadeghi, A.: Unsafe Exposure Analysis of Mobile In-App Advertisements. Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks 067(Section 2) (2012)
29. Hornyack, P., Han, S., Jung, J., Schechter, S., Wetherall, D.: These aren't the droids you're looking for: Retrofitting Android to protect data from imperious

applications. In: Proceedings of the 18th ACM conference on Computer and communications security. pp. 639–652. ACM (2011)
30. Jacobson, V., Braden, R., Borman, D., Satyanarayanan, M., Kistler, J., Mummert, L., Ebling, M.: RFC 1323: TCP extensions for high performance (1992)
31. Kohno, T., Broido, A., Claffy, K.C.: Remote physical device fingerprinting. Dependable and Secure Computing, IEEE Transactions on 2(2), 93–108 (2005)
32. Marlinspike, M.: New tricks for defeating SSL in practice. BlackHat DC, February (2009)
33. Murdoch, S.J.: Hot or not: Revealing hidden services by their clock skew. In: Proceedings of the 13th ACM conference on Computer and Communications Security. pp. 27–36. ACM (2006)
34. Soltani, A., Peterson, A., Gellman, B.: NSA uses Google cookies to pinpoint targets for hacking. https://www.washingtonpost.com/news/the-switch/wp/2013/12/10/nsa-uses-google-cookies-to-pinpoint-targets-for-hacking/ (2013)
35. Stevens, R., Gibler, C., Crussell, J.: Investigating User Privacy in Android Ad Libraries. IEEE Mobile Security Technologies (MoST) (2012)
36. Suarez-Tangil, G., Conti, M., Tapiador, J.E., Peris-Lopez, P.: Detecting targeted smartphone malware with behavior-triggering stochastic models. In: Computer Security-ESORICS 2014, pp. 183–201. Springer (2014)
37. Tekeoglu, A., Altiparmak, N., Tosun, A.Ş.: Approximating the number of active nodes behind a NAT device. In: Computer Communications and Networks (ICCCN), 2011 Proceedings of 20th International Conference on. pp. 1–7. IEEE (2011)
38. Tongaonkar, A., Dai, S., Nucci, A., Song, D.: Understanding mobile app usage patterns using in-app advertisements. In: Passive and Active Measurement. pp. 63–72. Springer (2013)
39. Vanrykel, E.: Passive Network Attacks on Mobile Applications. Master's thesis, Katholieke Universiteit Leuven (2015)
40. Vanrykel, E., Acar, G., Herrmann, M., Diaz, C.: Exploiting Unencrypted Mobile Application Traffic for Surveillance (Technical Report). https://securewww.esat.kuleuven.be/cosic/publications/article-2602.pdf (2016)
41. Weinstein, D.: Leaking Android hardware serial number to unprivileged apps. http://insitusec.blogspot.be/2013/01/leaking-android-hardware-serial-number.html (2013)
42. Wicherski, G., Weingarten, F., Meyer, U.: IP agnostic real-time traffic filtering and host identification using TCP timestamps. In: Local Computer Networks (LCN), 2013 IEEE 38th Conference on. pp. 647–654. IEEE (2013)
43. Xia, N., Song, H.H., Liao, Y., Iliofotou, M.: Mosaic: Quantifying Privacy Leakage in Mobile Networks. SIGCOMM '13 Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM (ii), 279–290 (2013)
44. Zander, S., Murdoch, S.J.: An Improved Clock-skew Measurement Technique for Revealing Hidden Services. In: USENIX Security Symposium. pp. 211–226 (2008)