# The Long-Standing Privacy Debate: Mobile Websites Vs Mobile Apps

Elias P. Papadopoulos[1,3], Michalis Diamantaris[1,3], Panagiotis Papadopoulos[1,3],
Thanasis Petsas[2,3], Sotiris Ioannidis[1,3], Evangelos P. Markatos[1,3]

[1]FORTH-ICS, Greece, {php, diamant, panpap, sotiris, markatos}@ics.forth.gr
[2]Appthority, USA, apetsas@appthority.com
[3]University of Crete, Greece

## ABSTRACT

The vast majority of online services nowadays, provide both a mobile-friendly website and a mobile application to their users. Both of these choices are usually released for free, with their developers, usually gaining revenue by allowing advertisements from ad networks to be embedded into their content. In order to provide more personalized and thus more effective advertisements, ad networks usually deploy pervasive user tracking, raising this way significant privacy concerns. As a consequence, the users do not have to think only their convenience before deciding which choice to use while accessing a service: web or app, but also which one harms their privacy the least.

In this paper, we aim to respond to this question: *which of the two options protects the users' privacy in the best way? apps or browsers?* To tackle this question, we study a broad range of privacy related leaks in a comparison of several popular apps and their web counterpart. These leaks may contain not only personally identifying information (PII) but also device-specific information, able to cross-application and cross-site track the user into the network, and allow third parties to link web with app sessions.

Finally, we propose an anti-tracking mechanism that enable the users to access an online service through a mobile app without risking their privacy. Our evaluation shows that our approach is able to preserve the privacy of the user by reducing the leaking identifiers of apps by 27.41% on average, while it imposes a practically negligible latency of less than 1 millisecond per request.

## 1. INTRODUCTION

### 1.1 The setting

As recently as only a decade ago, the only way to connect and interact with online services (or online web sites) was through *web browsers*. However, the proliferation of mobile devices and developer tools gave service providers the chance to create their very own mobile *applications* (or apps) that the users download and install in their devices. Each of the above two access choices (i.e. apps vs. web browsers), offers different kinds of advantages [57]. For example, web browsers can be found in most/all mobile devices and provide easy access to any mobile-friendly web site. On the other hand, native apps may offer better support for specific functionalities such as interactive gaming and offline access.

Choosing between the app and the browser is not easy. There is a lot of debating on the web, with several studies trying to compare these two options across different dimensions [57, 10, 17, 55, 64, 9]. Fortunately, the majority of service providers support both options: they provide *both* a mobile app *and* a mobile website. Although each option may have different benefits, in this paper, we are interested in exploring their privacy-related characteristics. That is, *which of the two options protects the users' privacy in the best way? apps or browsers?* Or similarly, *which of the two options facilitates the most privacy leaks?*

### 1.2 Related studies

Certainly, we are not the first ones to deal with this problem. For example, C. Leung *et al.* attempt a comparison of mobile apps and vanilla browsers based on the amount of personally identifiable information (PII) they leak [31]. The authors manually examined a small group of 50 online services and they monitored the PII each of them shares, over plaintext or encrypted connections. Their results show that there is no clear single answer. Therefore, they implemented an online service [32] aiming to recommend the users the best option for accessing a small sample of online services, based on the PII they care about the most.

In this paper, we conduct a similar comparison, but we significantly broaden the definition of privacy. Apart from personal data, such as gender, email address, name, username, birth-date, etc., that a service may leak, there is also device-specific information that can be used as identifiers. Such identifiers may include: (i) installed applications, (ii) known SSIDs, (iii) connected wifi, (iv) operating system's build information, (v) carrier, etc. These identifiers although seemingly unable, at a first glance, to reveal any possible sensitive data for the user, they are able, when combined, to allow a monitoring entity to persistently track mobile users without using any deletable cookies or resettable Advertising IDs [14, 63, 36]. This way, the monitoring entity can uniquely identify the mobile user and monitor her

*WWW 2017,* April 3–7, 2017, Perth, Australia.
ACM 978-1-4503-4913-0/17/04.
http://dx.doi.org/10.1145/3038912.3052691

behavior, her actions or her interests in the online world: information, which is usually more useful for the web entities (advertisers, analytics, etc.) to obtain than individual and possibly sensitive parts of personal data.

To make matters worse, by deploying device fingerprinting a third party can also: (i) decloak user's anonymous sessions: by linking for example Tor sessions of the same device with non anonymous ones [3, 59] and (ii) link web with app sessions, one of the biggest challenges of mobile ad networks [58]. Surprisingly, our results show that in case of device-specific privacy leaks, there is a clear winner: mobile browsers leak significantly less information compared to mobile apps. In most of the cases we studied, mobile apps leaked tons of information that mobile browsers did not (or could not) leak. Thus, we urge users to consider more seriously the use browsers whenever they have the choice.

Unfortunately, this choice may not always be available. For example, web sites may provide poor functionality to mobile devices and thus, the use of apps may seem the only reasonable choice from a user experience point of view. To improve the privacy of users, who *must* use mobile apps, we propose *antiTrackDroid*, an anti-tracking mechanism for mobile apps, tantamount to the current state-of-the-art ad-blockers of mobile browsers. Our approach constitutes an integrated monitoring and filtering module, which contrary to alternative approaches works solely in the users' device, without requiring any additional infrastructure (i.e. proxy or VPN). Our evaluation shows that *antiTrackDroid* is able to reduce the leaking identifiers of apps by 27.41% on average, when it imposes an insignificant latency of less than 1 millisecond per request.

To summarize, the contributions of this paper are:

1. We conduct a comparative study regarding the device-related privacy leaks of mobile apps and mobile browsers in Android. Our dataset includes a set of online services accessible by both mobile apps and mobile-friendly web pages. For each of them, we investigate if the associated app leaks more or less information than its website, accessed through mobile browser.

2. We design *antiTrackDroid*: a novel anti-tracking mechanism for mobile devices. Similar to state-of-the-art browser ad-blockers, our approach blocks any possible request may deliver to third parties data that can be used either for user profiling or device fingerprinting.

3. We implement our system as an integrated filtering module for Android. *antiTrackDroid* uses a mobile-based blacklist, which we publicly release, and it does not require changes in the respective OS or any kind of external infrastructure (i.e. proxy). We experimentally evaluate our prototype and show that it is able to reduce the leaking identifiers of apps by 27.41% on average, when it imposes an insignificant latency of less than 1 millisecond per request.

## 1.3 Background

**Third party tracking in web sites.** Traditionally, web sites keep track of users and sessions by using *cookies*. So by storing some state (in the form of cookies) on the client side, an advertising or analytics company can identify a user along with her interests, preferences, or even past purchases. The need for a more centralized infrastructure, which will work as a data warehouse containing rich data for several individual users and their interactions with different web sites, brought in the surface mechanisms like web beacons [35] and cookie synchronization [2]. These mechanisms allow third parties to collect user data bypassing the same origin policy [60], thus enriching their profiles of the users.

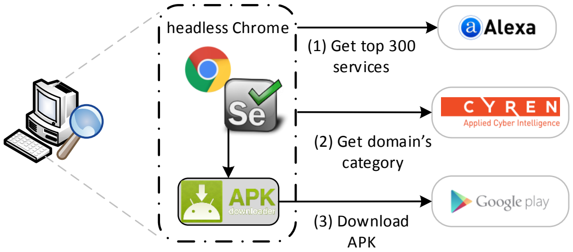


**Figure 1: High level overview of the data collection process.**

**Third party tracking in mobile apps.** To earn ad revenue, app developers display ads from third parties in their apps. These ads are usually delivered through RTB auctions [19]. To deliver effective advertisements to the end-user, third party ad-networks request from app developers to embed in their apps external third party ad-libraries, also known as in-app libraries [13]. The scope of these ad-libraries is to allow the developer request at runtime an ad-impression to fill the app's available ad-slots. To facilitate the delivery of personalized advertisement, ad-libraries inherit all the permissions enjoyed by the original app such as: access to the phone, access to contacts list, access to device characteristics, etc. In this way, ad-libraries can track users as they use services in cyberspace.

## 2. OUR DATASET

Our dataset contains several popular online services along with their mobile application (app) counterpart. We started with the 300 top online web services from Alexa and, for each one of them, we tried to find its corresponding mobile app. To automate the mobile apps collection process, we used the Selenium suite [49] to instrument Chrome browser and the APK downloader plugin [44] to download the full APKs of the Android apps from Google Play. Figure 1 summarizes our dataset collection process. Note that at the time of experimentation (February 2016) only 116 (of the top 300 Alexa sites) provided a mobile app. Thus, our final dataset consists of 116 apps along with their associated mobile-friendly web counterpart, making it larger than datasets explored in similar manually investigative studies([31]: 50 apps, [45]: 100 apps , [66]: 110 apps).

Before we analyze the privacy leaks of apps and web in our dataset, we first explore the characteristics of our dataset.

**Application Categories.** As presented in Figure 1, we used CYREN [12] to extract the category of each service in our dataset. Figure 2 shows the breakdown of our services into different categories. As expected, News-, Shopping- and Social-related services dominate the dataset, but it seems that we have services from all over the spectrum.

**Third party in-app libraries.** As we discussed in Section 1.3, the majority of free apps embed a third party, in-app library. These third party libraries are used for analytics- and ad- related purposes, thus sending to third parties information about the user, which is important for the tar-

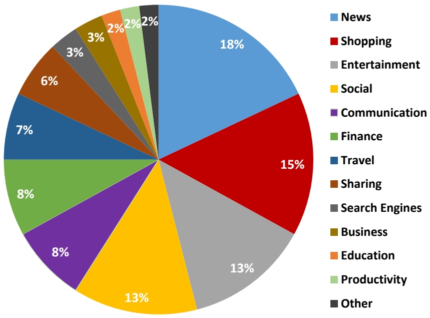


**Figure 2: Classification of apps based on the different content categories.**

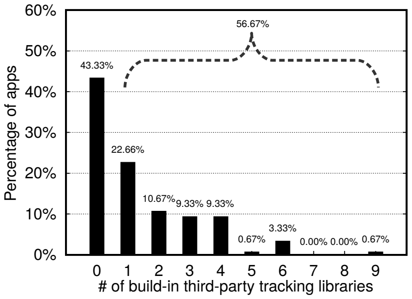


**Figure 3: Number of analytics- or ad-related libraries per app. 43.33% of apps does not contain any such library with the remaining 56.67% containing at least one.**

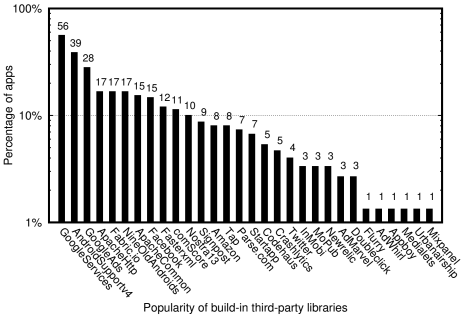


**Figure 4: Percentage of apps, in which each ad-libraries is detected.**

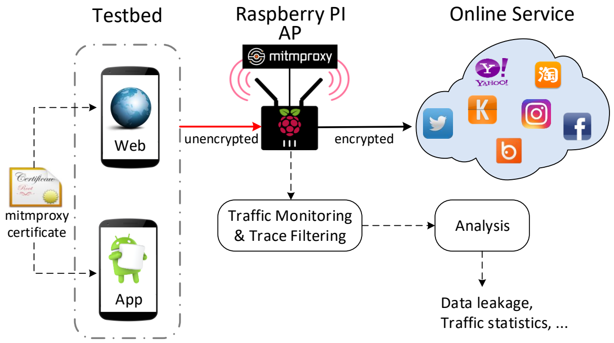


**Figure 5: Overview of the monitoring methodology of apps and web related traffic.**

geted advertising delivery. To identify these in-app libraries in our dataset, we use LibRadar [34], a tool for Android which detects embedded libraries, even if obfuscation methods were deployed. After the in-app library extraction, we manually filtered out libraries other than analytics- and ad-related. Figure 3 presents the analytics- and ad-related libraries found per app and as we see, 56.67% of apps contain at least one such library, when there is also 1 app with 9 of such in-app libraries. Note that these results, are verified by other studies as well [38]. In Figure 4, we see the popularity of the top third party libraries in our dataset, with GoogleAds residing in 28% of the apps and Fabric.io following with 16.67%.

## 3. MONITORING OUTGOING TRAFFIC

In Figure 5, we see an overview of our testbed. Using a NEXUS 6 smartphone running Android 6.0.1, we run each online service (i) from the corresponding app and (ii) from its website by using Firefox mobile browser[1]. Each run lasts for approximately 20 minutes where we perform the same user actions in both counterparts of the service including login, registration, search, share, etc. In order to capture the devices' network traffic (both HTTP and HTTPS), we used a dedicated monitoring component, which captures all the outgoing requests of both apps and browsers. For our monitoring component, we use a raspberry PI 2 [43] device configured as an access point and by running mitmproxy [11], an SSL-capable monitoring proxy, we are able to monitor SSL sessions as well. After capturing both HTTP and HTTPS traffic, the traces are forwarded to our *Traffic Monitoring & Trace Filtering* module, in which the tracking related requests are identified by using a filtering list based on a popular mobile-based blacklist [26], enhanced with entries we collected after manual inspection. Finally, the categorized traffic is passed to the *Analysis* module to produce statistics and the privacy leak analysis results. In this module, we filter possible leaked identifiers by performing pattern matching using a list of ID keywords we discover after studying device's settings. Moreover, we implemented an app able to collect all these IDs, with a view to find and verify the correctness of them. Then, we manually inspect the results to eliminate possible false positives.

**Muffling Background Apps.** To prevent apps running simultaneously during our experiments, we limit the background app activities while capturing the trace of each app in our dataset. More technically, we use the available developer options of the mobile device and by using a custom bash script that employs the adb toolkit [4], we kill the background processes of the device.

**Bypassing SSL Certificate Pinning.** Certificate (or SSL) Pinning [61, 23] is a technique used by several mobile apps to avoid MITM attacks. Through SSL pinning, a mobile app checks the certificate received by the server during the SSL handshake, and compares it to a known copy of the particular certificate, that was bundled with the app. To bypass SSL Certificate pinning and allow our monitoring component to capture the SSL traffic unimpeded, we use a modified version of JustTrustMe [62] module of Xposed Framework [47]. By using this module, we hook into SSL-related functions and nullify the code responsible for performing SSL pinning checks.

## 4. PRIVACY LEAK ANALYSIS

In this section, we present the core analysis of our privacy-related study. Specifically, by using the network traces we collected with our monitoring proxy, we measure and compare the quantity and the type of information leaked, as well

[1] We choose Firefox mobile browser for its ability to support browser extensions

| IDs | Description | PermissionGroup | App | Services(%) | Web | Services(%) |
|---|---|---|---|---|---|---|
| Build | The Android OS build version code | NONE | ✓ | 100.00 | ✓ | 0.00 |
| Model | The device model or its codename | NONE | ✓ | 100.00 | ✓ | 9.48 |
| OS Version | The OS or SDK version | NONE | ✓ | 100.00 | ✓ | 100.00 |
| Manufacturer | The device manufacturer | NONE | ✓ | 78.45 | ✓ | 24.14 |
| Screen Ressolution | The screen resolution of the device | NONE | ✓ | 75.00 | ✓ | 42.24 |
| Location | Device's GPS coordinates | LOCATION | ✓ | 66.38 | ✓ | 85.34 |
| Carrier | The Mobile Network Operator | PHONE | ✓ | 64.66 | - | 0.00 |
| Advertising ID | User-resettable, unique, anonymous ID for advertising, provided by Google Play services(ADID) | NONE | ✓ | 62.93 | - | 0.00 |
| Android ID | A random 64-bit number that is generated when the device boot's for the first time | NONE | ✓ | 57.76 | - | 0.00 |
| CPU | The device's CPU architecture | NONE | ✓ | 35.34 | ✓ | 20.69 |
| IMEI | International Mobile Equipment Identity | PHONE | ✓ | 24.14 | - | 0.00 |
| Timezone | User's timezone | NONE | ✓ | 24.14 | ✓ | 9.48 |
| City | The city name of the device's location | NONE | ✓ | 22.41 | ✓ | 25.86 |
| Device SN | A unique hardware serial number of the device | NONE | ✓ | 14.66 | - | 0.00 |
| MAC Address | The MAC address from the device WiFi NIC | WIFI STATE | ✓ | 14.66 | - | 0.00 |
| AP SSID | Access Point's MAC Address or SSID | WIFI STATE | ✓ | 9.48 | - | 0.00 |
| IMSI | International Mobile Subscriber Identity | PHONE | ✓ | 9.48 | - | 0.00 |
| Local IP | Device's local(LAN) IP address | WIFI STATE | ✓ | 6.03 | ✓ | 0.00 |
| Fingerprint | A string that uniquely identifies the device's build | NONE | ✓ | 5.17 | - | 0.00 |
| Memory Info | The device's (total/free) memory information | NONE | ✓ | 5.17 | - | 0.00 |
| Phone Number | The SIM number | PHONE | ✓ | 5.17 | - | 0.00 |
| WiFi Scan | Scan for nearby routers and devices and grab their MAC Address and SSID | WIFI STATE and LOCATION | ✓ | 4.31 | - | 0.00 |
| Contacts | The device's contacts list | CONTACTS | ✓ | 3.45 | - | 0.00 |
| Installed Apps | The device installed apps | NONE | ✓ | 3.45 | - | 0.00 |
| ICCID | The SIM card Serial Number | PHONE | ✓ | 2.59 | - | 0.00 |
| Kernel Version | The OS kernel version | NONE | ✓ | 2.59 | - | 0.00 |
| Baseband | The radio driver in which the info related to the telephone communications of the device is stored | NONE | ✓ | 1.72 | - | 0.00 |
| Bootloader | The system bootloader version number | NONE | ✓ | 0.86 | - | 0.00 |
| GSF | Google Services Framework Key ID, paired with the user's account | GSERVICES | ✓ | 0.86 | - | 0.00 |
| Stored SSIDs | The SSID/MAC of all connected Access Point's | WIFI STATE | ✓ | 0.86 | - | 0.00 |
| Logcat | The log of system messages, including stack traces | NONE | ✓ | 0.86 | - | 0.00 |
| SMS | The device's sent/received SMS | SMS | ✓ | 0.00 | - | 0.00 |

**Table 1: Description of each ID we investigate, their required permissions (Normal permissions are marked with blue, when Runtime/Dangerous permissions with red), their leakability by apps or browsers and the percentage of services found retrieving the corresponding value of each ID.**

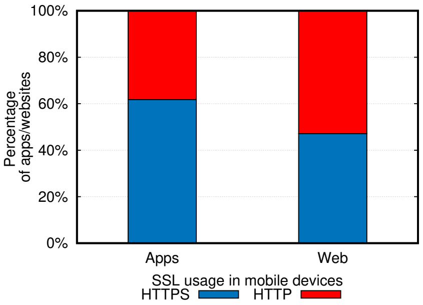


**Figure 6: Use of SSL in apps and web.**

as the diffusion of these leaks in both web and app versions of our collected online services.

Apart from the personal data a service may leak (such as birth-date, email addresses, gender, etc.), there is also device-specific information, which, if leaked, can also be used as identifiers. Although unable, at a first glance, to reveal any personal data for the user, these identifiers are able to allow a tracking entity to follow mobile users inside a network without using any deletable cookies or resettable Advertising IDs. Table 1 presents a short description of the leaks and identifiers we detected.

## 4.1 Encrypted sessions

Both apps and web browsers need to communicate with their associated online service in order to send and retrieve updated information. In our first experiment, we measure the adoption of SSL in the transactions of both apps and websites to explore the possibility of a passive observer to learn user info by monitoring the traffic.

First we measure the use of SSL in apps, and our results indicate that **only 18.97% of the apps use exclusively HTTPS**, 2.58% use solely HTTP, and 78.45% a susceptible [20] mixture of both. Consequently, it should not come as a surprise that we found 2 apps sending user's credentials in plaintext over HTTP. We informed the corresponding providers and we can confirm that at least one of them has fixed it. In Figure 6, we compare the use of SSL in web and apps. We see that **apps are more likely to use HTTPS (62% of total traffic) compared to web browsers (47%)**.

## 4.2 Identifiers leaked

In our next experiment, we set out to explore what kinds of identification information is leaked. Table 1 presents a list of such identifiers including "Advertising ID", "Android ID", "AP SSID", etc. along with its required permissions. We immediately see that apps are very aggressive at leaking such information (see column "Services(%))"). For example, we see that 57.76% of the apps leak the "Android ID" identifier. Surprisingly, we observe that none of the web browsers (see 4th column) leak this information. This happens, contrary to apps, because *web browsers typically do not have access to this information*. In summary, we see that **apps are much more prone than web browsers to leak device identification information**.

Table 1 also shows that mobile apps leak a huge variety of information including the list with the rest of the installed apps, too. This list allows an observing entity to easily infer important PII about the user including gender, age, preferences, interests etc. There are 3.45% of the applications, sending the whole list of the installed apps to a remote domain. Interestingly, in one of them, the remote server after conducting some analysis on the data, responded back with an approximation of the user's gender, age range, a list of possible interests and a number of recommended brand names.

In addition, there are also five applications and one website leaking the nearby WiFi Access Points. Such data can

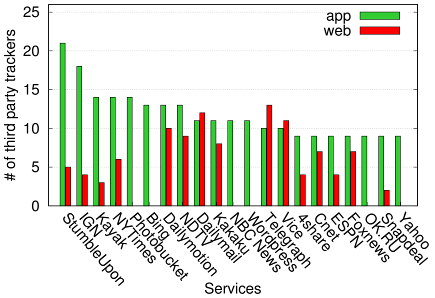


**Figure 7: Number of 3rd party tracking domains, with which the device interacts when the user accesses each of the top 20 privacy-leaking services.**

be correlated with online AP maps [7] and reveal the exact location of the user and possible interpersonal relations of people being in the same location at the same time. In addition, there is one app leaking the entire list of known APs, which can reveal previous locations the user has visited, even before the installation of the app. Finally, there is one app leaking the device's current running processes.

## 4.3 Diffusion of privacy leaks

After analyzing the type of information an app and a web browser can leak, we set out to explore the number of third party entities that receive this kind of information. First, we measure the number of third party trackers in apps and web browsers. Figure 7 shows that in most cases, it is the app that provides identifying information to more third party trackers. Indeed, we see that apps leak information to an average of 11.7 third party trackers while web browsers leak information to an average of 5 trackers. Similarly, as many as 93.9% of Android apps leak data to one or more third-party trackers, while the corresponding percentage for web browsers is 69.3%.

Finally, in Figure 8, we present the most popular tracking domains in the dataset of the collected apps. As we see, Google's analytics and advertising domains (google-analytics, doubleclick) dominate, having access in 56% and 53.6% of the apps respectively. Facebook, one of the most common social media apps, follows with 50.4%.

## 4.4 Mobile browsers leak too

Up to this point, we compare app and web counterparts of the online services. However, let us not overlook that mobile websites are being accessed by web browsers, which are mobile apps themselves. As a consequence, browser apps may leak data to remote entities as well. To explore this, we experimented with the 15 most popular web browsers in Android. We fetch a simple, tracker-free website like `google.com` using each one of them and we monitor their traffic. As we see in Figure 9, the vast majority of browsers sends a significant number of third party tracking requests to the network. Surprisingly, we see that **even the Adblock browser sends a request to a tracking domain** (i.e. `adjust.com`).

In order to further-investigate the possible privacy leaks of web browsers, we analyze the content of the above tracking requests. In Table 2, we present the identifiers each browser leaks to both first and third parties. We see that there is a significant number of browsers leaking an abundance of identifiers (more than 10 in some cases).

| Leaked IDs | Adblock | APUS | Boat | Chrome | CM Browser | Dolphin | DU Browser | Firefox | Maxthon | Next Browser | Opera | Opera Mini | UC Browser | Yandex |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Coordinates | - | ✓ | ✓ | - | ✓ | ✓ | - | - | ✓ | ✓ | ✓ | ✓ | - | ✓ |
| City | - | - | ✓ | - | ✓ | ✓ | - | - | ✓ | - | - | - | - | ✓ |
| Timezone | - | - | - | - | ✓ | - | - | - | - | - | - | - | - | ✓ |
| Android ID | - | - | - | - | ✓ | ✓ | ✓ | - | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Advertising ID | ✓ | - | ✓ | - | ✓ | - | - | ✓ | - | - | ✓ | ✓ | ✓ | ✓ |
| IMEI | - | - | - | - | - | - | ✓ | - | ✓ | - | - | - | ✓ | - |
| IMSI | - | - | - | - | - | - | ✓ | - | - | - | - | - | ✓ | - |
| ICCID | - | - | - | - | - | - | - | - | ✓ | - | - | - | - | - |
| MAC Address | - | - | - | - | ✓ | - | - | - | ✓ | - | - | - | ✓ | ✓ |
| Device SN. | - | - | - | - | - | - | - | - | - | - | - | ✓ | ✓ | - |
| OS Version | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | - | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Build Version | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | - | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Carrier | - | - | ✓ | - | ✓ | ✓ | ✓ | - | - | - | ✓ | ✓ | ✓ | - |
| Manufacturer | ✓ | - | ✓ | - | ✓ | ✓ | ✓ | ✓ | - | - | ✓ | ✓ | - | ✓ |
| Model | - | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| CPU Arch | - | - | - | - | ✓ | - | - | - | - | - | ✓ | ✓ | - | ✓ |
| Screen Resol. | ✓ | - | ✓ | - | ✓ | ✓ | ✓ | ✓ | - | - | ✓ | ✓ | - | ✓ |
| AP SSID | - | - | - | - | - | - | - | - | - | - | - | - | - | ✓ |
| WiFi Scan | - | - | - | - | - | - | - | - | - | - | - | - | - | ✓ |
| **TOTAL** | 5 | 4 | 9 | 3 | 13 | 9 | 9 | 4 | 9 | 5 | 10 | 11 | 10 | 14 |

**Table 2: Identifiers leaked by the most popular browser apps when visiting `google.com`.**

A careful reader may have observed that although some identifiers are not leaked from a website (see Table 1), interestingly, they are getting leaked through the browser app itself (see Table 2). Hence, we see that mobile web browsers constitute ordinary apps, thus including third-party trackers of their own. As a consequence, when a user visits a website (e.g `CNN.com`), the website running inside the browser's sandbox cannot access, for example AdvertisingID, but the browser app (along with its included third-party trackers) can, pairing this way the website visit with the specific AdvertisingID.

## 4.5 Performance cost of user tracking

Trackers do not cost only in the privacy of the user, they also consume resources of both web and apps by generating requests not relevant with the content the user chose to browse. In Figure 10 and Figure 11, we present the number of the total and tracking requests respectively for both web and app versions of our collected online services. We see that apps, in general, send more requests to the network (367 for the 50% of the apps) than web (221 for the 50% of the websites), when the portion of tracking requests is 8.5% and 5% respectively. Regarding bytes, we see in Figure 12 and Figure 13, that the transferred volume of bytes is similar in both apps and web, as expected given their similar functionality. The tracking related bytes are 192 KB in apps and 77 KB in web per service. Apparently, this amount of Bytes regard unnecessary tracking content, constituting a significant monetary cost for the user's data plan.

## 4.6 Summary

In the above analysis, we explore the information leaked in each of the two versions of an online service. We see that both web and apps leak important fingerprinting information about the user's device. This allows third parties to not only cross-channel track the users by linking web with app sessions but also correlate eponymous with anonymous sessions. In addition, we see apps leaking information (e.g. installed and running apps, nearby APs, etc.) that may allow a tracking domains to infer the user interests, gender, even behavioral patterns. Furthermore, we studied the diffusion of the above privacy leaks and we see that apps tend

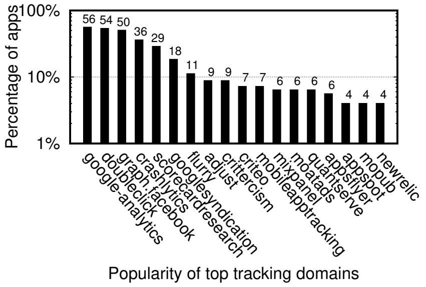


**Figure 8: Percentage of apps, in which the top tracking domains were detected.**

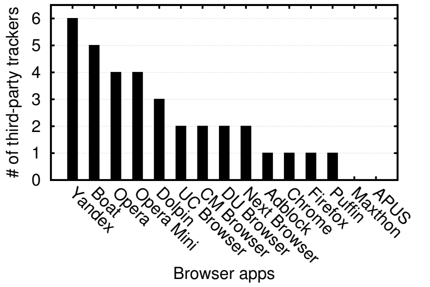


**Figure 9: Number of requests to tracking domains for each browser, while fetching `google.com`.**

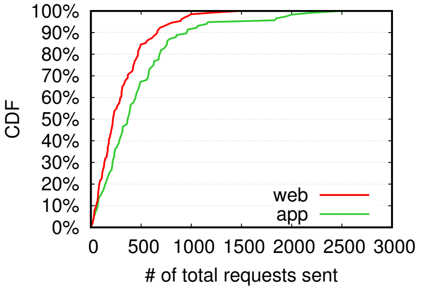


**Figure 10: Distribution of total requests sent.**

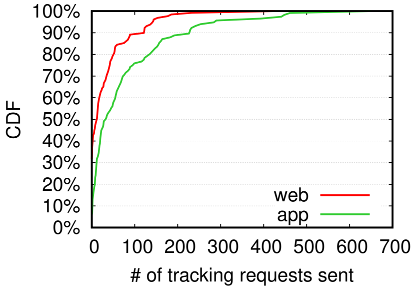


**Figure 11: Distribution of tracking requests sent.**

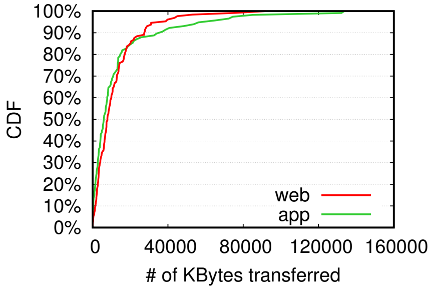


**Figure 12: Distribution of total KBytes transferred.**

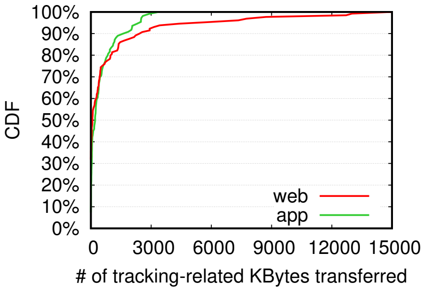


**Figure 13: Distribution of the tracking related KBytes transferred.**

to send requests to more tracking domains than their web counterpart.

Our results prove that both versions of the online service leak information that can be used beyond the control of users (for targeted advertising purposes or an asset for sale to other entities [29]). However, recalling the motivating question of our paper: *to identify which of the two, web or app, facilitates the most privacy leaks*, the answer is straightforward. Apps leak significantly more device-specific information.

# 5. FORTIFYING APPS FROM TRACKERS

## 5.1 Our approach: antiTrackDroid

Our findings so far suggest that apps leak more information than web browsers. Thus, it is reasonable for privacy-aware users to prefer using web browsers instead of apps to access online services. However, this is not always possible, or desired [57]. As a result, the use of mobile apps is, in many cases, unavoidable. To provide these users with better privacy guarantees, we propose *antiTrackDroid*: an anti-tracking mechanism able to preserve the privacy of the users by blocking many personal and device information leaks to any third parties. Specifically, antiTrackDroid is a module which filters all outgoing requests and blocks the ones delivering tracking information.

The core design principles of antiTrackDroid include the ability to operate (i) for all apps, and (ii) without the need for any additional infrastructure (e.g. VPN, Proxy, etc.). To meet these principles, antiTrackDroid leverages Xposed [47]: a popular Android framework, which allows system-level changes at runtime without requiring installation of any custom ROM or modifications to the application. By using Xposed, antiTrackDroid is able to intercept every outgoing request and check if the destination's domain name exist in a blacklist of mobile trackers. In case of match (i.e. the destination is blacklisted), the outgoing request is blocked. Figure 14 summarizes the design of our approach.

## 5.2 Implementation

To assess the effectiveness and feasibility of our approach, we implemented a prototype of antiTrackDroid for Android. Our system consists of the following main components:

1. The *Filtering module*, which implements the `IXposedHookLoadPackage` and filters the tracking requests based on the Xposed module.

2. An *Android Activity* (hereafter named `Launcher`), with a graphical user interface to allow the users configure the Filtering module.

3. The *AppList Updater*, which listens for newly added or removed packages and updates the list of applications being monitored, using a Broadcast Receiver.

**Launcher Activity.** `Launcher` acts as an interface between the Filtering module and the user. It contains a menu allowing the user to (i) load a different blacklist or exclude an application from the filtering procedure. `Launcher` is also responsible of maintaining two different data structures: a HashSet with the tracking domain names loaded from the blacklist, and a HashSet with the applications being monitored. By using HashSets, antiTrackDroid is able to perform look-ups with O(1) complexity reducing significantly the per request latency overhead.

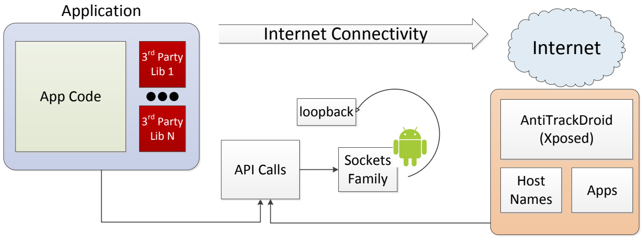


**Figure 14: Defense mechanism overview.**

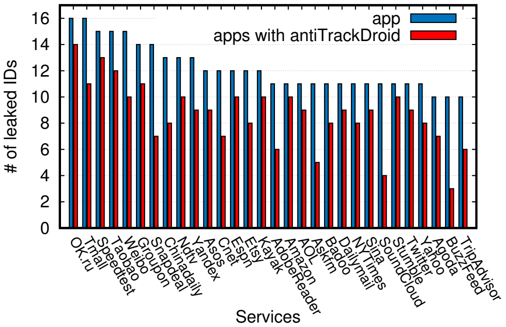


**Figure 15: Number of leaked ID without and with antiTrackDroid for the 30 apps with the higher number of ID leaks.**

**Filtering Module.** Mobile applications send data over HTTP/HTTPS requests by using TCP sockets. Therefore, Filtering module dynamically hooks on the constructor of the TCP socket opened by the applications residing in the HashSet of monitored apps. In addition, it re-writes the destination IP address with `localhost` in case of a blocked request. This loop-back interface is a virtual network interface that does not correspond to any actual hardware, so any packets transmitted to it, will not generate any hardware interrupts. By redirecting to loop-back, antiTrackDroid avoids possible crashes of apps caused by aborted connections.

**AppList Updater.** Since users may install or remove applications at any time, our system must be able to update the list of monitored apps. In Android, every time an app is added or removed in the system, a broadcast message is send through the `PackageManager` component, which can reach any app in the device. By using a `Broadcast Receiver` [5], the AppList updater, running as a background service, can listen such messages and update the list of monitored apps.

**Blacklist of Trackers.** To determine if a request is a tracker or not in the *Filtering Module*, we use the popular mobile-based blacklist of AdAway [26], which we extended by including the tracking domains we collected manually during our privacy leak analysis. Our publicly released blacklist of antiTrackDroid which we update frequently, currently contains 66k entries in total. Recall that in Launcher Activity, the user is free to change the used blacklist by loading one of her choice.

# 6. EVALUATION

In this section, we evaluate the effectiveness and performance of our antiTrackDroid, and we explore its benefits.

## 6.1 Privacy performance

To evaluate the privacy preservation of antiTrackDroid, we inspect the identifiers leaked to the network with and without the use of antiTrackDroid. In Figure 15, we see the number of leaked IDs with and without antiTrackDroid for the 30 more leaking apps. Our results show that antiTrackDroid is able to reduce the number of leaked identifiers by 27.41% on the average. Note that since our approach blocks the majority of third party trackers, the rest of the leaking IDs exist due to requests destined to the developer's first party domains and content providers (e.g. CDNs). Blocking such requests would cause degradation of the user experience or even fatal error to the application.

## 6.2 Latency overhead

Although antiTrackDroid significantly improves privacy, it may have an impact on the overall latency of the apps as well. Indeed, antiTrackDroid may increase latency because it includes an extra check with the blacklist. On the other hand, it may significantly reduce the latency imposed by blacklisted tracker requests as these requests will be blocked and the app will not have to suffer their latency. To measure the impact on latency, we created an Android app, with which we can send arbitrary number of requests to a server of ours. Thus, we create 1000 requests carrying 15KB of data each, and we send these requests to the server sequentially after a short time interval. We run this experiment 3 times: (i) once with antiTrackDroid switched off, (ii) once with antiTrackDroid enabled and the domain not included in the blacklist (benign request), and (iii) one more with antiTrackDroid enabled and the domain of the server blacklisted (Tracker request).

Figure 16 shows that the vanilla (no antiTrackDroid) request (see 1st bar) takes about 100 ms. When we switch on antiTrackDroid (see 2nd bar), the latency is practically the same. Indeed, a few lookups in a blacklist do not add any overhead noticeable in the 100 ms range (less than 1 ms). Finally, when we switch on antiTrackDroid and make an access to a tracker (see 3rd bar), the latency drops to less than 10 ms as the request is blocked. We are happy to see that antiTrackDroid, not only improves privacy, but it also improves performance.

## 6.3 Benefits from the use of antiTrackDroid

Besides preserving the user's privacy, the blocking functionality of our approach improves also the performance of apps in the user's data-plan and battery.

**Bytes transferred.** By blocking the tracking related requests antiTrackDroid is able to save a significant amount of data, an aspect of great importance when it comes to mobile users with specific data plan. To determine the different volume of data transferred to/from the apps in a device running antiTrackDroid, we conduct the following experiment: we run all apps in our device as previously, but instead of blocking the requests we calculate the outgoing bytes of requests and the incoming bytes of the associate responses. In addition, we measure the overall traffic of the app and finally calculate the portion of traffic marked as tracker-related. Figure 17(a), presents the results, where we see that antiTrackDroid reduces the volume of transferred bytes by 8% for the 50% of the apps.

**Energy cost** There are several studies [24, 37] attempting to measure the energy cost imposed by the ad-related content to a user's device. It is apparent that every connection an app opens with a network entity, it imposes an overhead

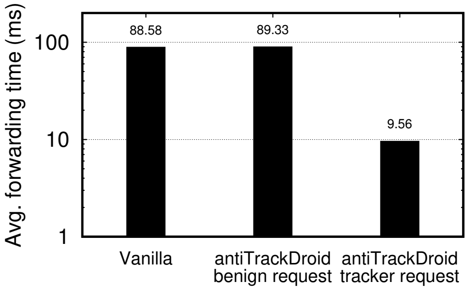
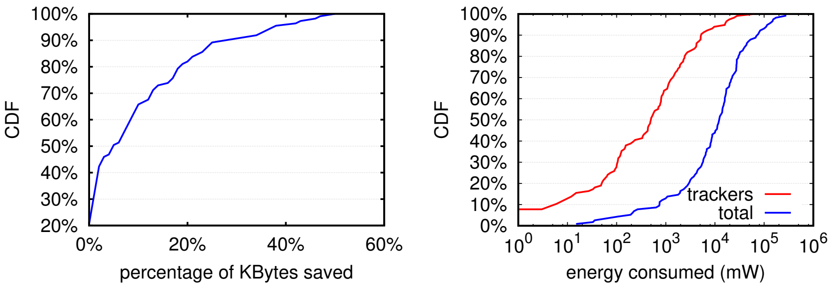


**Figure 16: Overall request forwarding time with and without antiTrackDroid.**



(a) Percentage of KBytes saved.

(b) Watts saved due to the less tracking request sent.

**Figure 17: Benefits from the use of antiTrackDroid.**

to the overall energy consumption of the device [39]. As a consequence, by reducing the requests an app sends or receives, along with their transferred data, antiTrackDroid is able to reduce the energy cost of the application as well.

Measuring the energy consumption in a mobile device is a challenging task. In order to estimate our gain with antiTrackDroid, we perform a simulation, based on the energy readings of Appscope [65]. Figure 17(b) presents the distributions of the per-app power consumption for (i) the total and (ii) the tracker-related transferred bytes. From our simulation we find that there is a significant reduction of about 7,5% for the 50% of the applications.

## 7. RELATED WORK

There are numerous studies aiming to explore the privacy leaks of browsers and apps, without though attempting a direct head-to-head comparison to help users decide what is better for their privacy.

### 7.1 Privacy leak measuring studies

Roesner and Kohno in [46] propose a taxonomy to classify tracking behaviors beyond the traditional notions of first- and third-party tracking. There is also a significant body of research that studied privacy leaks and user fingerprinting in web-based online services. [41, 14, 2, 28, 6, 15], when some of them propose also countermeasures [27, 42, 40]. However, these works are focused solely on desktop browsers; mobile browsers constitute a different ecosystem, where different ad companies and analytics dominate, retrieving even more tracking information for both user and device. In addition, the content of the websites is also different with less available ad-space and less capabilities in the web developers toolchest [16].

On the other hand, there are various notable studies regarding privacy leaks in mobile applications. Leontiadis et al., for example, in [30], analyze a large dataset of 250,000 Android apps and their results reveal the ineffectiveness of current privacy protection mechanisms. Finally, they propose a market-aware privacy protection framework aiming to achieve an equilibrium between the developer's revenue and the user's privacy. Senevirante et al. [50] conduct a large-scale study comparing the presence of tracking libraries in paid and free apps. Their key findings denote that almost 60% of paid apps, contain trackers that collect sensitive information, compared to 85%–95% found for free apps.

In [66], Zang et al. conduct a survey on over a hundred Android and iOS apps to study what kind of information leaks to third-party domains. They dynamically analyzed the apps by running each for 10 to 20 minutes, while performing manually testing, with valid user data and performing reasonable UI interactions. They found that, 73% of Android and 43% of iOS apps shared sensitive information with third parties. Han et al. [25], performed a real-world tracking study of mobile apps running on the devices of 20 participants instrumented with dynamic taint tracking of specific sensitive information. They found that 36% of the domains perform user tracking and that 37% of them use persistent identifiers that allow cross-application and cross-site profiling of the user. Grace et al. in [22], studied 100,000 apps and found that ad libraries are embedded in 52.1% of the apps. The behavior of some of these ad-libraries were suspicious monitoring of the user's call logs, phone number, the applications installed on the phone, or even her browser bookmarks. Book et al. [8], studied 100,000 apps over a long period of time. They showed that the use of permissions in ad libraries has increased over time. In [53], the authors study the mobile ad isolation in popular Android adSDKs, and investigate what types of information a mobile advertising entity can learn about the user of the device Demetriou et al. [13], explored the capabilities of various ad libraries and showed that ad networks are prone to leak more of the user's data than before. Therefore, they propose Pluto, a mobile risk assesment framework to discover personal data leaks. In [33], the authors propose a method to automatically locate personal information (PI) e mbedded in network traffic to Internet services. Unfortunately, their approach is based on inspection of data exchanged by Internet services, so it works only with unencrypted connections.

Similar to our work, Stevens et al. [56], analyzed 13 popular Android ad libraries. Their findings indicate privacy violations compared to in-browser advertising. The ad libraries take advantage from permissions not declared in the ad library documentation, but are needed by the apps. In another similar study [31], the authors compare mobile apps and web browsers based on the amount of demographic information they leak. The authors manually examined a small group of 50 online services and monitored the PII each of them leaks. In our paper, we extend the investigated privacy leaks to include device specific identifiers as well. Our results show that by broadening the spectrum of leaks, web facilitates less leaks than apps.

## 7.2 Proposed countermeasures

There are also several attempts aiming to solve the problem of leaking sensitive information to third-party domains in mobile devices. In [51], for example, Shekar et al. present an extension that splits application's functionality code and ad code to run in separate processes, thus eliminating the ability of applications to request extra permissions on behalf of their ad libraries. AdAway [48] is an ad blocker available for Android. It comes in the form of an app which requires root access and is based on a blacklist implemented using the `hosts` file. Contrary to AdAway, our countermeasure does not require any Read/Write access on the device's system partition, something that is not applicable for specific devices. Moreover, any third-party library could easily read the host file if it contains their domain(s) (no root permission needed), hence it could simply change the domain to one not presented in the host file.

Recon [45], is a VPN-based system capable of identifying PII leaks through the network. By combining machine learning and network trace analysis, Recon is able to completely block or add noise to PII leaking requests. A similar VPN-based approach is AntMonitor [52]: a crowd-sourcing system, which leverages Android's SDK VPNService to detect sensitive data leakage on Android applications. However it detects and prevents leakage of sensitive information only over unencrypted traffic. PrivacyGuard [54] also uses VPNService to monitor private data leak and it is also able to randomize the outgoing data to protect the user's privacy. Unfortunately, it sends fake data not only to third-party trackers, but to first-parties as well, thus risking the application's seamless functionality. Contrary to the above approaches, our implementation does not need any trusted entity (e.g. VPN or Proxy), to monitor the users traffic. Besides, there are also countries, which currently ban or block VPN-related traffic [1].

Towards a different direction, in [21] Gordon et al. implemented DroidSafe: a static information flow analysis tool that detects and reports potential leaks of sensitive information in Android applications. The authors tested their approach against 24 real world applications.

# 8. CONCLUSION

The proliferation of mobile apps give the users the option to choose between the associated app or web browser counterpart to access an online service. Each one of them has its own advantages, but what about the user's privacy? In this paper, we attempt to identify which harms the least the user's privacy: web browsers or apps? Specifically, we conducted an extensive study of a broad spectrum of privacy-related leaks able to expose not only PII but also device specific identifiers. These identifiers allow third parties to deploy fingerprinting mechanisms, that (i) track the user into the network (ii) link web with app session and (iii) correlate anonymous (such as tor) sessions with eponymous ones. Our study suggests that apps leak more information than web browsers.

Finally, to help users improve their privacy even when using a mobile app, we propose *antiTrackDroid*: an anti-tracking mechanism for Android apps, similar to the state-of-the-art ad-blockers of mobile browsers. Evaluation results show that our approach is able to reduce the privacy leaks by 27.41%, when it imposes a negligible overhead of less than 1 millisecond per request.

**Future Work:** iOS owns a large share of the market, therefore we plan to extend our privacy leak study to iOS, and compare our findings in Android. This way, we will be able to determine which of the two platforms facilitates the most privacy leaks. Moreover, we aim to extend our defense mechanism as well to fortify apps from third party trackers in iOS. This can be done by leveraging Cydia Substrate [18], a framework which allows developers to provide run-time patches to system's functions.

# 9. ACKNOWLEDGMENTS

This work was supported by the project SHARCS, under Grant Agreement No. 644571. In addition, this project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 690972. The paper reflects only the authors' view and the Agency is not responsible for any use that may be made of the information it contains.

# 10. REFERENCES

[1] Using VPN in the UAE? You'll Be Fined Up To $545,000 If Get Caught! http://thehackernews.com/2016/07/vpn-is-illegal-in-uae.html.

[2] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz. The web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, 2014.

[3] G. Acar, M. Juarez, N. Nikiforakis, C. Diaz, S. Gürses, F. Piessens, and B. Preneel. Fpdetective: Dusting the web for fingerprinters. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer &#38; Communications Security*, CCS '13, pages 1129–1140, New York, NY, USA, 2013. ACM.

[4] Android Developers. Android Debug Bridge. http://developer.android.com/tools/help/adb.html.

[5] Android Developers. Class Overview: BroadcastReceiver. `http://developer.android.com/reference/android/content/BroadcastReceiver.html`.

[6] M. D. Ayenson, D. J. Wambach, A. Soltani, N. Good, and C. J. Hoofnagle. Flash cookies and privacy ii: Now with html5 and etag respawning. *SSRN 1898390*, 2011.

[7] bobzilla, arkasha, and uhtu. Wigle: Wireless geographic logging engine. https://wigle.net/.

[8] T. Book, A. Pridgen, and D. S. Wallach. Longitudinal analysis of android ad library permissions. *CoRR*, 2013.

[9] C. Borodescu. Web sites vs. web apps: What the experts think. https://www.visionmobile.com/blog/2013/07/web-sites-vs-web-apps-what-the-experts-think, 2013.

[10] I. Brodsky. Deathmatch: The mobile web vs. mobile apps. http://www.computerworld.com/article/3016736/mobile-wireless/the-mobile-web-vs-mobile-app-death-match.

[11] A. Cortesi. An interactive console program that allows traffic flows to be intercepted, inspected, modified and replayed. https://mitmproxy.org/, 2015.

[12] CYREN. Cyren – Cloud-based Internet Security Solytions. http://commtouch.com/.

[13] S. Demetriou, W. Merrill, W. Yang, A. Zhang, and C. A. Gunter. Free for all! assessing user data exposure to advertising libraries on android. In *NDSS*, 2016.

[14] P. Eckersley. How unique is your web browser? In *International Symposium PETS*, 2010.

[15] S. Englehardt, D. Reisman, C. Eubank, P. Zimmerman, J. Mayer, A. Narayanan, and E. W. Felten. Cookies that give you away: The surveillance implications of web tracking. In *Proceedings of the 24th WWW*, 2015.

[16] M. Firtman. Html5 compatibility on mobile and tablet browsers with testing on real devices. http://mobilehtml5.org/, 2015.

[17] R. Fishkin. Mobile web vs mobile apps: Where should you invest your marketing? https://moz.com/blog/mobile-web-mobile-apps-invest-marketing-whiteboard-friday.

[18] J. Freeman. Cydia Substrate: The powerful code modification platform behind cydia., 2008. http://www.cydiasubstrate.com/.

[19] A. Ghosh and A. Roth. Selling privacy at auction. In *Proceedings of the 12th ACM Conference on Electronic Commerce*, pages 199–208, New York, USA, 2011. ACM.

[20] Google Developers. Mixed content weakens https. `https://developers.google.com/web/fundamentals/security/prevent-mixed-content/what-is-mixed-content#mixed_content_weakens_https`, 2017.

[21] M. I. Gordon, D. Kim, J. H. Perkins, L. Gilham, N. Nguyen, and M. C. Rinard. Information flow analysis of android applications in droidsafe. In *NDSS*, 2015.

[22] M. C. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi. Unsafe exposure analysis of mobile in-app advertisements. In *Proceedings of the Fifth ACM WISEC '12*, 2012.

[23] J. Graves. Ssl pinning for increased app security. https://possiblemobile.com/2013/03/ssl-pinning-for-increased-app-security/, 2013.

[24] J. Gui, S. Mcilroy, M. Nagappan, and W. G. J. Halfond. Truth in advertising: The hidden cost of mobile ads for software developers. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ICSE '15, pages 100–110, Piscataway, NJ, USA, 2015. IEEE Press.

[25] S. Han, Jaeyeon Jung, and David Wetherall. A study of third-party tracking by mobile apps in the wild, 2012.

[26] Kicelo and D. Schuermann. Adaway default blocklist. https://adaway.org/hosts.txt, 2016.

[27] G. Kontaxis, M. Polychronakis, A. D. Keromytis, and E. P. Markatos. Privacy-preserving social plugins. In *Proceedings of the 21st USENIX Conference on Security Symposium*, Security'12, pages 30–30, Berkeley, CA, USA, 2012.

[28] B. Krishnamurthy, D. Malandrino, and C. E. Wills. Measuring privacy loss and the impact of privacy protection in web browsing. In *Proceedings of the 3rd Symposium on Usable Privacy and Security*, SOUPS '07, New York, 2007.

[29] S. Kroft. The data brokers selling your personal information. http://www.cbsnews.com/news/the-data-brokers-selling-your-personal-information/, 2009.

[30] I. Leontiadis, C. Efstratiou, M. Picone, and C. Mascolo. Don't kill my ads!: Balancing privacy in an ad-supported mobile application market. HotMobile, 2012.

[31] C. Leung, J. Ren, D. Choffnes, and C. Wilson. Should you use the app for that?: Comparing the privacy implications of app- and web-based online services. In *Proceedings of the 2016 ACM on Internet Measurement Conference*, IMC '16.

[32] C. Leung, J. Ren, D. Choffnes, and C. Wilson. App vs web. https://recon.meddle.mobi/appvsweb/, 2016.

[33] Y. Liu, H. H. Song, I. Bermudez, A. Mislove, M. Baldi, and A. Tongaonkar. Identifying personal information in internet traffic. In *Proceedings of the 2015 ACM on Conference on Online Social Networks*, pages 59–70. ACM, 2015.

[34] Z. Ma, H. Wang, Y. Guo, and X. Chen. Libradar: Fast and accurate detection of third-party libraries in android apps. In *Proceedings of the 38th ICSE*, 2016.

[35] D. Martin, H. Wu, and A. Alsaid. Hidden surveillance by web sites: Web bugs in contemporary use. *Commun.*, Dec. 2003.

[36] C. C. Miller and S. Sengupta. Advertisers find new ways to track smartphone users. `http://www.bostonglobe.com/news/nation/2013/10/05/selling-secrets-phone-users-advertisers/ZSNNChJQvFuEcHJFsUJGUM/story.html`, 2013.

[37] P. Mohan, S. Nath, and O. Riva. Prefetching mobile ads: Can advertising systems afford it? In *Proceedings of the 8th ACM European Conference on Computer Systems*, EuroSys '13, pages 267–280, New York, NY, USA, 2013. ACM.

[38] M. Nagappan. Go ahead and add that extra ad library, but be careful about which one you add. https://www.developereconomics.com/add-extra-ad-library-but-be-careful-which-one, 2015.

[39] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. Munafò, K. Papagiannaki, and P. Steenkiste. The cost of the ”s” in https. In *Proceedings of the 10th ACM CoNEXT*, 2014.

[40] N. Nikiforakis, W. Joosen, and B. Livshits. Privaricator: Deceiving fingerprinters with little white lies. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, pages 820–830, Republic and Canton of Geneva, Switzerland, 2015. International World Wide Web Conferences Steering Committee.

[41] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, Washington, DC, USA, 2013. IEEE Computer Society.

[42] P. Papadopoulos, A. Papadogiannakis, M. Polychronakis, A. Zarras, T. Holz, and E. P. Markatos. K-subscription: Privacy-preserving microblogging browsing through obfuscation. In *Proceedings of the 29th Annual Computer Security Applications Conference*, ACSAC '13, pages 49–58, New York, NY, USA, 2013. ACM.

[43] Raspberry Pi Foundation. Raspberry Pi 2 Model B. https://www.raspberrypi.org/products/raspberry-pi-2-model-b/.

[44] redphx. Apk Downloader. https://chrome.google.com/webstore/detail/apk-downloader/cgihflhdpokeobcfimliamffejfnmfii.

[45] J. Ren, A. Rao, M. Lindorfer, A. Legout, and D. Choffnes. Recon: Revealing and controlling pii leaks in mobile network traffic. In *Proceedings of the 14th Annual International Conference MobiSys*, 2016.

[46] F. Roesner, T. Kohno, and D. Wetherall. Detecting and defending against third-party tracking on the web. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation 2012.*

[47] rovo89. Xposed Module Repository. http://repo.xposed.info/.

[48] D. Schurmann. Adaway: An open source ad blocker for android using the hosts file. https://adaway.org/.

[49] Selenium. Selenium – Web Browser Automation. http://www.seleniumhq.org/.

[50] S. Seneviratne, H. Kolamunna, and A. Seneviratne. A measurement study of tracking in paid mobile applications. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, WiSec '15, 2015.

[51] S. Shekhar, M. Dietz, and D. S. Wallach. Adsplit: Separating smartphone advertising from applications. In *Proceedings of the 21st USENIX Security*, 2012.

[52] A. Shuba, A. Le, M. Gjoka, J. Varmarken, S. Langhoff, and A. Markopoulou. Antmonitor: Network traffic monitoring and real-time prevention of privacy leaks in mobile devices. In *Proceedings of the 2015 Workshop on Wireless of the Students, by the Students, & for the Students*. ACM, 2015.

[53] S. Son, D. Kim, and V. Shmatikov. What mobile ads know about mobile users. In *23nd Annual NDSS*, 2016.

[54] Y. Song and U. Hengartner. Privacyguard: A vpn-based platform to detect information leakage on android devices. In *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*, 2015.

[55] G. Sterling. Morgan stanley: No, apps aren't winning. the mobile browser is. http://marketingland.com/morgan-stanley-no-apps-arent-winning-the-mobile-browser-is-144303.

[56] R. Stevens, C. Gibler, J. Crussell, J. Erickson, and H. Chen. Investigating user privacy in android ad libraries. In *Workshop on Mobile Security Technologies (MoST)*, 2012.

[57] J. Summerfield. Mobile website vs. mobile app: Which is best for your organization? https://www.hswsolutions.com/services/mobile-web-development/mobile-website-vs-apps/.

[58] TRUSTe Technology Blog. Mobile tracking: How it works and why it's different. http://www.truste.com/developer/?p=86, 2016.

[59] UnhappyGhost Goldenstein. Fingerprinting defenses in the tor browser. http://www.unhappyghost.com/2015/02/forensics-fingerprinting-defenses-in-tor-browser.html.

[60] W3C Web Security. Same origin policy. `https://www.w3.org/Security/wiki/Same_Origin_Policy`, 2010.

[61] J. Walton, JohnSteven, J. Manico, K. Wall, and R. Iramar. Certificate and Public Key Pinning. `https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning`.

[62] R. Welton. Android SSL certificate pinning bypass. https://github.com/Fuzion24/JustTrustMe.

[63] M. Whitener. Cookies are so yesterday; cross-device tracking is in-some tips. https://iapp.org/news/a/cookies-are-so-yesterday-cross-device-tracking-is-insome-tips/.

[64] L. Wroblewski. Mobile web vs. native apps or why you want both. http://www.lukew.com/ff/entry.asp?1954, 2016.

[65] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha. Appscope: Application energy metering framework for android smartphone using kernel activity monitoring. In *Proceedings of the 2012 USENIX ATC.*

[66] J. Zang, K. Dummit, J. Graves, P. Lisker, and L. Sweeney. Who knows what about me? a survey of behind the scenes personal data sharing to third parties by mobile apps. http://techscience.org/a/2015103001, 2015.