Marcos Tileria* and Jorge Blasco

Watch Over Your TV: A Security and Privacy Analysis of the Android TV Ecosystem

Abstract: The rapid adoption of Smart TVs has resulted in them becoming another app-based ecosystem. In this context, Android TV is one of the major players as it is widely available across multiple TV manufacturers and has a high integration with other Google products. Yet, the Android TV ecosystem has remained unexplored. This paper presents a deep analysis of the Android TV ecosystem using a large dataset of TV apps. We give an insight into the stakeholder ecosystem, including developers, streaming services, and third-party libraries. We analyze the behavior of TV apps in terms of sensitive data collection and communication with other devices using a pipeline of static analysis tools, network traffic collection, and verification via manual analysis. We compare the mobile and TV version of popular streaming apps and found a significant degradation of TV apps in terms of quality and different data collection practices. Our study shows that most TV apps present potentially harmful behaviors, and in most cases, these can be attributed to tracking and advertisement services. We found a prevalence of static identifiers for tracking purposes despite this not being the recommendation. This finding suggests that Google's new policies limiting advertising identifiers will not have a tangible effect on the TV ecosystem.

Keywords: Smart TV, privacy, data leakage, tracking, advertising

DOI 10.56553/popets-2022-0092
Received 2021-11-30; revised 2022-03-15; accepted 2022-03-16.

***Corresponding Author: Marcos Tileria:** Royal Holloway, University of London, E-mail: Marcos.Tileria.2016@live.rhul.ac.uk
Jorge Blasco: Royal Holloway, University of London, E-mail: Jorge.BlascoAlis@rhul.ac.uk

1 Introduction

In recent years, Internet-based TV has gained popularity to the detriment of traditionally broadcast TV model [9]. The popularity of streaming services, affordable prices, and enhanced user experience have contributed to the rise of the Smart TV market, which is expected to reach $193 billion by 2021 [67].

In addition to Smart TVs, it is possible to use an OTT (over-the-air) device to convert almost any device with a screen and HDMI port into a Smart TV capable of delivering content over the Internet. Smart TVs rely on TV apps or channels to deliver streaming content to users. However, TV apps can also be used to play games, browse the web, and other general-purpose utilities.

Smart TVs offer enhanced capabilities and connectivity in comparison to traditional TVs. However, the additional capabilities also introduce vulnerabilities [1, 36, 46, 51, 68] and potential privacy violations [14, 17, 49]. Users' Personal Identifiable Information (PII) such as unique identifiers, hardware address, or viewing habits can be exposed to TV providers and analytics/advertising services.

A big part of the Smart TV business relies on analytics and advertising [45, 57]. In a similar way to mobile apps, TV apps can embed third-party libraries for these purposes. Media providers use personal data to personalize video content, and advertisers use personal data to better target ads to the viewers. Some companies provide both services, streaming via a Content Delivery Network (CDN) and analytics services. Despite this heavy dependency on personal data, many users are unaware that TV apps can access and share their data with third parties, even when most consider such practices unacceptable [41].

Recent works [45, 57] have focused on traffic analysis and the detection of tracker domains on the Roku and Amazon Fire platforms. In this paper, we are interested in measuring the Android TV ecosystem and understanding the threats to user's security and privacy. We selected the Android TV ecosystem because of its availability across TV brands and high integration with

Android, which facilitates consumer adoption but also access to their applications for analysis. In this context, we want to identify privacy-invasive behaviors in TV apps and identify the stakeholders behind such practices. Additionally, we want to analyze the interaction of TV apps with other devices and their relationship with tracking and advertising services. In particular, we make the following contributions:

- We provide a detailed study of the Android TV ecosystem. We collected more than 4.5k TV apps from different markets. The dataset is four times larger than previous evaluations [45, 57], contains streaming apps and a myriad of TV-compatible utility apps and games that have proven to be equally invasive regarding sensitive data collection. Additionally, we compare the TV and mobile version of streaming popular apps and present main differences between them.
- We develop a static analysis pipeline for TV apps analysis complemented with traffic analysis experiments and manual analysis. Our framework also includes a novel categorization of sources and sinks in the Android and TV ecosystem. We find that most TV apps present potentially harmful and privacy-invasive behaviors and that tracking and advertising libraries are responsible for a significant part of these cases.
- By analyzing inter-device communication patterns in TV apps, we found that developers typically rely on old APIs to communicate with other devices and do not implement best practices to secure communication channels when using new APIs. We observe that around 50% TV apps include the code to open ports; this number is significantly greater than previous reports in the mobile ecosystem [34, 69].
- We find malware and apps with invasive behavior in our dataset. We responsibly disclose our results so security and privacy can be improved in the app ecosystem. We notified Google of applications having abusive practices not included in their privacy policies. We also notified developers using non-secure practices so they can fix their apps and make them more secure.

We make our framework publicly available[1], along with our results, so that other researchers can benefit from this data.

1 https://gitlab.com/s3lab-rhul/watch-over-your-tv-paper

2 Background

Android TV is a Smart TV platform developed by Google on top of Android OS. Android TV has been designed to be highly coupled with Android. In other words, TV apps follow the same implementation pattern of other platforms such as Android mobile. TV apps offer an additional abstraction that enables the interaction with media content from the Internet and Smart TV hardware.

2.1 TV Apps

TV apps are similar to mobile apps as they possess the same structure and use the same languages and development tools. Code and resources are packed into a single file (APK), although Google is enforcing a new publishing format in the Play Store from August 2021[2]. The most important difference with other platforms resides in the user interaction. Users are expected to watch TV from medium distances, and the input is based on a directional pad and a select button.

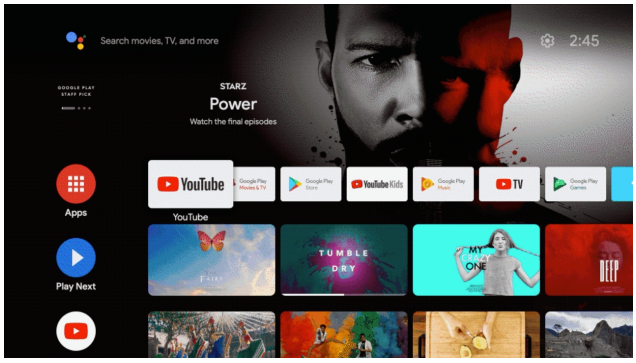

Fig. 1. Android TV Home screen

Another difference is that Smart TV users interact with the apps through the Home screen. The Home screen (Figure 1) is the Android TV's main interface that provides access to apps, content recommendations, and global search. Users can access apps directly via the Apps menu or by searching channels or programs that apps add to the Home screen. In Figure 1, each row is

2 The Bundle format does not affect our results as users still install APKs on their devices https://android-developers.googleblog.com/2021/06/the-future-of-android-app-bundles-is.html

a channel containing cards for each available program. When a user selects a program, the Home screen interacts with the corresponding app.

TV apps require custom configurations in the APK Manifest to run on Android TV [25]: 1) The APK must not declare unsupported hardware such as a touchscreen. 2) It must declare a launcher TV activity. 3) It can optionally declare support of the `Leanback` library that provides user interface templates, paging, and other features that are exclusively for Android TV.

TV apps deliver media content or provide standard utilities. Like other Android apps, TV apps have access to the file system, sensors, and network connections. However, developers are restricted by the limitations of the underlying hardware, e.g., no precise geolocation due to the lack of GPS chipsets or telephony services.

The TV Input Framework (TIF) is a set of libraries that facilitates the interaction of TV apps with media source providers [25]. Examples of TV apps implementing the TIF are Netflix, HBO, and Disney+. The TIF offers an interface to build apps emulating the TV broadcast style. It specifies channels, programs, track information, sessions, and other components required to display media content in a TV broadcast style.

2.2 Communications

TV apps can establish connections with other hosts. For instance, a TV app can exchange data with a remote server or with a device connected to the same network or in close proximity. For this, developers can use plain sockets [16, 34, 69] or high-level Android APIs like the Nearby platform [6, 27].

The Nearby platform is available for proximity communication since Android 4.0. It provides the `NearbyConnection` and `NearbyMessage` APIs to communicate with nearby devices that are not required to be connected to the same network. The `NearbyConnection` API offers the capability to discover and connect with other devices using multiple protocols, while the `NearbyMessage` provides a publish/subscribe communication model. Similarly, the `WifiDirect` API allows two devices to communicate even if the two are not connected to the same network. The Cast-TV API allows mobile apps to display content on a Smart TV, but this interaction is handled internally by Android TV.

2.3 Security Model and Threats in Android TV

Android TV apps are subject to the same security model as Android mobile apps: each app runs on a sandbox with minimum permissions by default. Android permissions are broadly divided into normal and dangerous. Normal permissions are granted automatically at installation time, and users can grant dangerous permissions at run time. Additionally, manufacturers and developers can create custom permissions to protect their apps' resources or share them with other apps.

As with regular mobile apps, developers of TV apps can use third-party libraries (TPL) to add services or functions to their apps. Such services include analytics, advertising and social networks, among others. Libraries inherit all permissions from the host app, which might give them access to sensitive data. Previous works reported multiple TPL collecting sensitive data such as unique identifiers, geolocation, and user data [12, 37]. More concerning, developers might be unaware of these data collection practices and other threats to the user's privacy [52].

While the Android apps ecosystem has been extensively studied, previous studies on Smart TV applications have focused on applications available for other platforms such as Roku and Amazon Fire TV [45, 57][3]. In this work, we analyze the Android TV ecosystem to provide the community with a better understanding of the security and privacy risks of Android TV and compare the risks of these apps with their mobile counterparts. Because of this, our work does not describe new threats within the ecosystem but specifically focuses on the TV environment and, in particular, the following issues:

– Bad development practices. This includes incomplete or false information in signing certificates and permission misuses, such as inconsistent definitions caused by porting code across platforms. Over-privileged apps request more permissions than they need [10, 70] and are problematic because they increase the attack surface on Smart TVs, amplify the impact of bugs and vulnerabilities, and violate the principle of least privilege [63]. Additionally, design shortcomings of custom permissions can lead to privilege escalation attacks by exploiting duplicate names vulnerabilities [38].

3 While Amazon Fire TV is heavily based on Android, their apps cannot make use of all the APIs available on Android TV.

– Sensitive data leakage. Particularly, PII collected by third-party services. TV apps also have access to personal data specific to Android TV, for instance, the title and genre of programs defined in the TIF. This information about users' viewing behavior can be used for user profiling [51, 57, 73]. A previous study shows that more than 70% of Smart TV users consider it unacceptable that advertisers access their viewing history [41].
– Vulnerable inter-device communications. This interaction poses security and privacy implications for consumers [1, 51]. Smart TV users are exposed to eavesdroppers listening to vulnerable communications that can be exploited [44].
– Malicious applications. TV app can present malicious behavior. This can be due to developers adding malicious code or malware exploiting some vulnerability [6, 11].

3 Dataset Collection and Characterization

Identifying and collecting TV apps at scale is challenging because the official Android market does not provide platform-specific metadata (TV, Mobile, Wear). Therefore, we use a initial seed of well-known TV apps to search related apps in the Play Store. We collect all identified package names and then download the last version of each app (as of August 2020) from AndroZoo [4], a well-known repository of Android apps widely used in the literature [3, 47, 65, 66] that includes a VirusTotal evaluation summary. We inspect the manifest of all downloaded apps to verify they can be run on Android TV. We also search for TV apps in APKMirror [7] using filters and pattern matching and download them with a crawler. In total, we collected 4745 TV apps with unique package names from both repositories.

	Touchscreen disable	TV Activity	TV Libraries	TIF
TV-only	√	√	√	×
TV-enabled	√	√	×	×
TV-streaming	√	√	√	√
Mobile-streaming	×	×	×	×

Table 1. Categorization of apps included in our dataset.

We classify the 4745 TV apps in four categories according to the configurations shown in Table 1:
– `TV-only`. This group includes apps that can run exclusively on Smart TVs. They require: 1) touchscreen disabled 2) declare a TV activity 3) include libraries that are specific to Android TV (Leanback).
– `TV-streaming`. The primary purpose of these TV apps is to show streaming content. These apps require the same configuration as the TV-only apps, plus they implement the TV Input Service. We inspect the APK bytecode to detect this feature.
– `TV-enabled`. These apps can run on Smart TVs or other devices like smartphones. Apps in this category only require to declare a TV activity and the touchscreen disabled.
– `Mobile-streaming` includes mobile apps whose primary purpose is to show streaming content, for instance, the Netflix mobile version. These apps do not require any setting from Table 1.

It is worth mentioning that we assign a TV app in the `TV-streaming` category if the app implements the TIF. Although the TIF is the standard way to communicate with a media provider, it is possible that some developers use a custom implementation for this interaction.

Table 2 shows a summary of the apps in the dataset. Even though our focus is on supposedly benign TV apps, there are 34 TV APKs that are flagged as malware in VirusTotal by more than 5 antivirus engines, of which 60% are considered malicious by more than 10. We use the malware threshold similar to other works [3, 47]. We expand the discussion of malware in Section 6.

TV-only	TV-enabled	TV-streaming	Mobile-streaming
831	3911	100	90

Table 2. Dataset classification. Note that one TV app can be in the TV-only and TV-Streaming group at the same time.

4 Ecosystem Overview

This section provides an overview of the ecosystem of Android TV, which includes developers, permissions, and third-party libraries, including tracking, and advertising services. Our analysis is based on features extracted via static analysis.

4.1 Developers

We start the ecosystem analysis by examining the developers behind the apps signing certificates. This information allows us to attribute apps to the same developer and identify the ones that incur heavily on privacy invasive behavior. While self-signed certificates cannot be trusted as an identity source, they can be used to identify apps that have been developed by the same organization.

First, we use Androguard [21] to extract the SHA256 fingerprint from the certificates to identify unique developers. Then, we use the field `Issuer` to attribute one certificate to its organization. In total, we identify 3623 unique fingerprints. 75% of the developers own only one app in our dataset. In the remaining group, 50% publish two apps and only 12% more than five apps. This trend shows that most developers publish one or two TV apps, similar to other studies [20, 62].

Figure 2 shows developers with multiple apps sorted by number of apps. Notably, the name in the Issuer field tends to differ from the real name of the organization or the name published in the Play Store. Most developers have apps from the same category, e.g., games or radio. Although there are some exceptions like the developers "Andromo" and "Android". Andromo is an app-builder platform which suggests that apps built with this platform are signed with a certificate owned or generated by the platform. The developer "Android" is not related to Google, and it uses different names such as Italy-games, Play Station Mobile Inc, and Google Commerce in the Play Store.

Considering the `TV-streaming` category, there are only 4 developers with multiple apps. Likewise, developers from the `TV-only` group tend to publish a single app, and developers with multiple apps correspond almost exclusively to the game category. We further investigate if there are developers in the `Mobile-Streaming` and `TV-only` and found that the groups are largely disjoint with only 3 developers in both groups.

Looking at the supposedly `Country` field of the certificates, we detected that United States organizations sign 47% of the APKs, following by China (6%), Canada (4%), and the United Kingdom (3%). Interestingly, we found 85 APKs without information about the signing organization, including empty fields or completed with irrelevant strings. 1234 apps use the same information as the Google apps excluding the signature. We check this list against the apps published by Google in its Play Store profile and determine that Google indeed develops only 30 of these apps. These results show that the bad practice of using irrelevant information, already reported in the mobile ecosystem [20, 53], it is also present in the Android TV ecosystem.

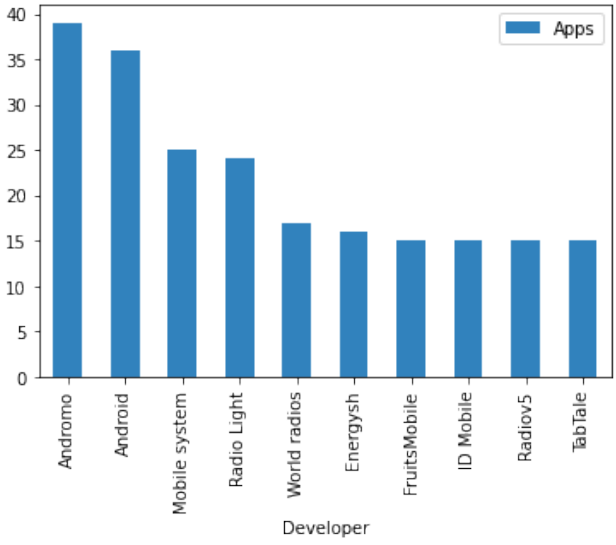

Fig. 2. Top 10 developers. Names correspond to the ones found in the certificate which may not relate to the name of the company behind the certificate as they are self-signed.

4.2 Third-Party Libraries

In this section, we analyze the presence of third-party libraries (TPL) in our dataset. Identifying TPL allows us to distinguish between data exposure to first/third parties, where the latter represents a more significant risk to user's privacy [43, 57]. Additionally, it allows us to characterize tracking and advertising services in the Android TV ecosystem.

First, we use Libscout [12], a library detection tool that is resilient to common obfuscation techniques to search the signature of embedded libraries in the APKs. Libscout relies on a repository that contains 402 unique library profiles with more than 8K signatures considering all versions[39].

Second, we implemented a lightweight analysis using Androguard to detect libraries not present in the Libscout repository. Our approach consists of several steps: 1) We disassemble each APK, extract all package name prefixes, and filter out irrelevant packages, e.g., well-known Android packages. 2) We cluster the packages according to the prefixes, and we keep the ones that appear in at least 10 apps. 3) We add the libraries to our repository, and we manually classify them using the categories from previous works [12, 37] and Privacy

Grade [31]. Note that some libraries might fit in more than one category, but we decided to assign one category per library. We discuss the limitations of our approach in Section 7.

Table 3 summarizes the 843 TPL libraries identified in our dataset, of which 64 are not present in other well-known repositories [12, 37]. We detected Social Media libraries in 88% of the apps, with multiple Facebook libraries occupying the top 5. Similarly, we found that 75% of the apps contain analytics libraries and 77% contain advertising libraries. 66% of the apps include advertising, analytics, and social media libraries together. Our analysis detected cloud libraries in 223 APKs, most of them owned by Amazon.

Category	# Libs	# Apps	Examples
Advertising	47	3637(77%)	Ogury, Appodeal
Analytic	16	3595(76%)	Firebase-analytics
Cloud	23	210(4%)	Amazon-Kinesis-Streams
Social Media	10	4195(88%)	Facebook, Twitter, VK
Utilities	747	4591(97%)	OkHttp, Stetho
Total	843	4745	

Table 3. Third-party library summary

We focus particularly on analytics and advertising libraries given recent ad spending spikes in Android due to changes in iOS privacy policies [13]. In this context, Google Play Services ads (2502 APKs) and Firebase analytics [4] (2407 APKs) are the most popular libraries. Other ad libraries found include Vungle, Tapjoy, and Jirbo. Smaato and FreeWheel are platforms specialized in video advertising and connected TV services. In total, we detected 70 tracking and advertising libraries, many of which were linked to data violation policies in the past [18, 58], some examples are Appodeal (84 APKs) and Umeng (27 APKs). Our static analysis show that a large number of data flows correspond to these libraries and we were able to confirm these findings with network traffic analysis experiments (more details in Section 5).

We also detected atypical libraries such as Javassist (11 APKs) that could affect the accuracy of static analysis. This library can modify Java bytecode and obfuscate Android apps, which is the case in some TV apps that we manually analyzed, e.g., `com.funkidslive.action`. Similarly, previous studies found that the game engine Unity reads and uploads hardware addresses to a remote server using covert channels [50] and persistently requests information about installed apps [52]. Our static analysis results indicate that this library might be collecting similar information in 278 APKs.

4.3 Permissions

Permissions specify the system-protected resources that TV apps are allowed to access. In this section, we analyze Android (AOSP) and custom permissions, focusing on user's privacy and user awareness.

We leverage Androguard to extract and analyze permissions from the TV apps. We identify a total of 225 AOSP permissions and 2600 custom permissions. Although the number of permissions per app varies greatly, Table 4 shows the average and maximum permission request per type. Next, we show data about the permission usage to understand the current state in our dataset.

	AOSP		Custom	Total
	Normal	Dangerous	-	-
avg	6.05	2.1	4.7	12.9
max	91	20	123	234

Table 4. Permissions summary

AOSP Permissions. Figure 3 shows the top 10 dangerous and normal permissions requested by TV apps. As one might expect, permissions related to network connectivity are at the top of the list. TV apps request 6 normal and 2 dangerous permissions on average. Although we focus on the use of dangerous permission, we discuss in section 6 some aspects of normal permission.

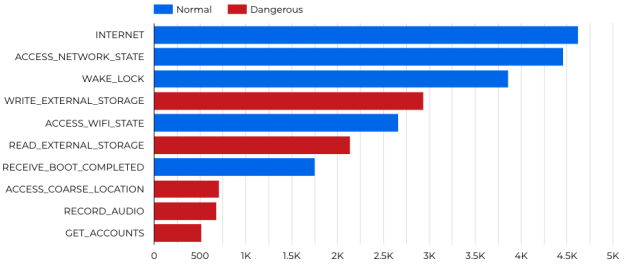

Fig. 3. Top 10 normal and dangerous permissions

Dangerous Permission. These permissions protect the most sensitive system resources. The most requested

4 Renamed Google analytics 4

dangerous permissions are related to external storage, location, record, audio, and account information (see Figure 3). We note that multiple TV apps request permissions not available in Android TV, such as telephony-services, and sensors. This peculiarity suggests that developers are porting mobile apps to TV without considering security aspects, leading to inconsistent apps.

User Consent. We analyze if TV apps are following the guidelines when requesting dangerous permissions. In Android, the API `shouldShowRequestPermissionRationale` returns a boolean indicating whether the app should show UI with rationale before requesting a dangerous permission. The official Android developer guidelines indicate that the app should explain the reason for requesting a permission and the effects of the user denying the request [24].

Therefore, we implemented a lightweight static analysis to detect if developers explain the rationale behind a permission request. Our analysis searches in the bytecode: 1) strings with the permission value, 2) API methods to request permission, for instance `requestPermissions` 3) the API method `shouldShowRequestPermissionRationale` and UI methods to show dialog, e.g., alerts, banners. Then we infer if the app is showing a rationale by matching the context of the results.

Table 5 shows the results for the top 5 dangerous permissions found in our dataset. Our results indicate that only a small percentage of TV apps display the rationale behind dangerous permission and that most developers are not following the guidelines. While our work has limitations due to the usage of the static analysis, our results are in line with a recent work that showed that only 8% of mobile apps provide extra information when requesting permission using dynamic analysis [19]. The authors of this work also argue that timing might be a better strategy than showing a rationale (e.g., requesting the GPS permission just before accessing the location). In fact, previous works show that people are more careful in giving consent when they are presented with more information about data collection and sharing behaviors [5, 56].

We manually check 10 popular TV apps to verify if they show this rationale. In half of the cases, we detected potentially over-privileged apps where we could not find the functionality associated with a particular permission, e.g., the app `com.mlbam.wwe_asb_app` requests the microphone permission, but we did not find such option. Note that TV apps UI are much simpler than mobile apps, which makes manual exploration more feasible. In the successful cases, we only observed the default request and no rationale. Some examples are `com.ted.android.tv` and `com.disney.disneyplus` that request the `record_audio` permission. Additionally, the default request tells the user that this permission can be revoked in the app's settings, but we could not find such option.

Dangerous Permissions	#Apps	# show context
ACCESS_COARSE_LOCATION	686	22 (3%)
GET_ACCOUNTS	492	6 (1%)
RECORD_AUDIO	672	37 (5%)
READ_EXTERNAL_STORAGE	2091	62 (1%)
WRITE_EXTERNAL_STORAGE	2881	186 (6%)

Table 5. TV App showing rationale when requesting dangerous permissions

Custom Permissions. We identify 2600 unique custom permissions requested across all APKs. 1809 permissions are declared by apps present in our dataset, while external apps declare the remaining 791 permissions. After using pattern-matching, searching in the web, and matching with the list of third-party libraries (Section 4.2), we were able to attribute 522 permissions to well-known developers such as Google, Amazon, Nvidia, HTC, and 22 permissions to third-party libraries. It is worth noting that naming convention is not enforced in Android which can result in attribution mismatch. However, these heuristics give us a notion of custom permission owners.

Considering `TV-enabled` apps, around 35% declare and 90% request at least one custom permission. 87% are signature-level permissions, and the remaining part is divided between mixed, normal, and dangerous in this order. Previous studies give the number of apps using custom permissions between 25% and 65% [38, 55]. We attribute this difference to the target apps included in this study (Android TV). Our work shows that Android TV apps normally require resources from other apps to work.

Interestingly, most apps do not complete the description field with meaningful information. The description field should describe the behavior in case a permission is granted or denied [26]. Over 94% of the apps leave this field empty, including four apps declaring dangerous custom permissions showing how most developers ignore this policy.

We found multiple cases of custom permissions with duplicated names, e.g., the permissions[5] READ_EXTERNAL_STORAGE (AOSP), and MANAGE_DOCUMENT (custom), declared by 2 and 6 apps, respectively. Duplicated permissions open the space to confuse deputy attacks and custom permission upgrade attacks [55] since the second permission is ignored by the OS [26]. Moreover, this scenario can lead to dangling and inconsistent permissions that can be exploited by attackers [38].

Particular Cases. We found 34 apps that request the custom permission ACCESS_SUPERUSER. This permission, although currently deprecated, is declared by a well known app (Superuser) to allow users to get root privileges. Most apps requesting this permission are file managers and operating system utilities. However, we also found streaming apps (`com.mobzapp.camstream`), TV remote control (`com.cetusplay.remoteservice`), screen cast (`com.mobzapp.screencast`), and screen recorder apps (`com.mobzapp.recme.free`) requesting it. At the moment of writing this paper, most of the apps still declare this permission.

The permission QUERY_ALL_PACKAGES protects the list of packages installed in a device. This information was unprotected before Android 11. The documentation specifies a policy for acceptable use of this permission, which includes device search, antivirus, file managers, and browsers [23]. We found 10 APKs using this permission, of which all but one meet the policy[6]. `com.ultimateguitar.tabs` is an app to learn how to play the guitar and other musical instruments, and it is not clear why this app requires access to this type of information as it is not mentioned within their privacy policy[7].

5 Behavioral Analysis

The Android TV ecosystem analysis provides intuitive information about specific app features such as permissions and their libraries. However, this information is not enough to determine if TV apps are capable of disseminating sensitive information or using vulnerable communications channels. In this section, we use static analysis tools to analyze these threats, and we verify some of our results with dynamic analysis and network captures.

5.1 Intra-Device Analysis

Static Analysis. We use taint tracking to detect potential sensitive data leaks. This analysis gives information about the propagation of data from a sensitive sources to a sensitive sink. We use Flowdroid [8] because it provides a good balance between accuracy and performance on real-world apps [15, 48]. We customized Flowdroid by adding sources and sinks that are specific to Android TV. The sources include media metadata, identifiers, and other methods added in the latest Android releases. Additionally, we added callbacks from the Leanback library and other media-related libraries to enable the creation of a more accurate control flow graph.

Dynamic Analysis. The traffic analysis experiment executes TV and mobile apps on real devices and then analyzes their network traces, searching for identifiers and user-sensitive data. We rely on Charles [60], an HTTP proxy, to intercept and decrypt network traffic. We instrumented the APKs using the mitm-proxy script [32] to add Charles certificate and remove certificate pinning checks. This instrumentation is only necessary for TV apps as there is no way to install custom certificates on Android TV. For the mobile apps, we use smartphones with Android 6 where the apps trust user-installed certificates. We created testing credentials to log in on each app (if possible) as previous work [57] showed that apps contact more third-party domains after login. Then we explore the app for 15 minutes trying to stream content and trigger ads.

5.2 Experimental Results

We run the static analyses on the entire dataset of 4745 APK files. For the dynamic analysis, we downloaded 65 popular streaming APKs directly from the Play Store; we called this group `Streaming-popular`. This group contains 1) 25 pairs of Mobile-TV apps (50 APKs). 2) 15 APKs compatible for both platforms. Additionally, we also use these APKs to compare the TV/mobile version of popular streaming apps. Note that we only run the traffic analysis on the `Streaming-popular` group as manual exploration does not scale, mainly because we decided to log in on each app during the testing and the

5 android.permission omitted in the name.

6 The enforcement of this policy was postponed until March 2022.

7 While reporting this to Google we found out that an update released on 11 Dec 2021 removed this permission request.

exploration timeout. The list of apps can be found in the Appendix.

5.2.1 Static Analysis

We run Flowdroid with a timeout of 8 minutes per analysis. The experiment was conducted on a server with 46 cores Intel Xeon CPU E5-2697 v3 @ 2.60GHz, and 92 GB of memory. The analysis failed for a small number APKs. 29 timed out after doubling the original time limit. In our experience, analysis that surpasses this timeout runs for hours. Another 18 failed for other reasons such as errors while parsing APK resources or unexpected bytecode.

The analysis found at least one sensitive data flow in 78% of the files. The number of sensitive data flows varies greatly: for instance, 313 APKs contain only one result. In contrast, 470 APKs contain more than 50 sensitive flows, and on the extreme side, we could observe 78 APKs having more than 100. The percentage of sensitive data flows per app category is as follows: TV-only (81%), TV-enabled (75%), TV-streaming (83%), and Mobile-streaming (92%).

Figure 4 shows data flows for a selected number of sources. We limit the output due to space constraints and to help with visualization. Full results are shown in tables 6 and 7. The results show that a large percentage of sensitive flows happen in TPL, including categories such as Tracking_ID (52%), Hardware_ID (45%), and Wireless_ID (48%).

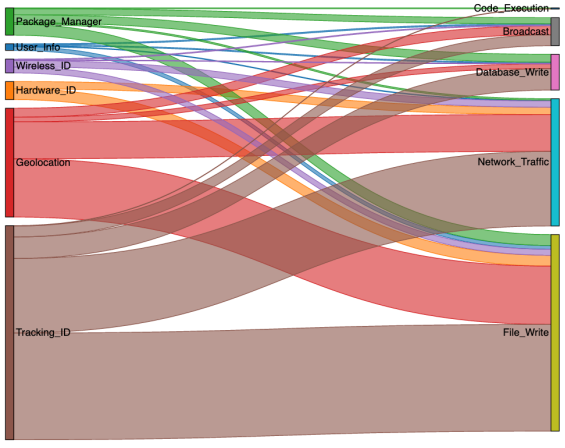

Fig. 4. Sankey diagram of sensitive data flows

Note that many flows reach non-Internet sinks. Even though these channels do not necessarily indicate data leaks, they could expose sensitive data to a second data flow. We were able to detect 1398 extra data flows using SharedPreferences and File API methods as connectors. However, understanding indirect data flows requires modeling each potential connector and more complex analysis. Moreover, we found data flows in 1256 APKs logging sensitive data, of which 80% correspond to TPL. One example is the Kochava SDK logging UUIDs, SSID, and geolocation in apps like Sling and NBC Sports. It is important to mention that logging sensitive data is discouraged by Google because of risk exposure and performance [28].

Now, we focus our discussion on selected categories that are common targets of abusive and malicious developers and the prevalence of trackers.

Identifiers. It is well known that tracking and profiling are pervasive in the Android ecosystem [14, 20, 49, 63]. We investigate what identifiers are being used by TV apps by checking the the data flows sources. The most used identifier is a globally unique ID (GUID) generated with the java.util.UUID package (3031 APKs). The most common sinks for this source are Logs (48%), Shared_Preferences (33%), File_Write (8%), and Network_Traffic (5%). The sources are collected by app's component only in 10% of the cases while advertisement libraries are responsible for 72% these data flows.

We highlight that resettable identifiers are largely absent from our results. In contrast, we found 285 APKs with data flows reading MAC-Address and SSID identifiers, of which around half of these cases correspond to TPL, despite the Google recommendation of not using hardware identifiers for tracking purposes [29]. Most of the APKs (88%) with static identifiers data flows share this data with at least one third-party library. For instance, the Facebook SDK has data flows in 58% of these apps, Tapjoy 11% and Presage 4%.

User Private data. From the category `User_Info` the most requested sources are the list of accounts, followed by display-name and email. The most popular sink categories are Logs (66%), File_Write and SharedPreferences (11% each), and a few instances of Network_Traffic and system Broadcast. Around half of the cases correspond to information flows in third-party libraries. 725 APKs (18%) contain information flows from the `Package_Manager`, including the list of installed packages (169 APKs). 83% of these calls are made from TPL, of which AppsFlyer (47%) and Bytedance (11%) are the most noticeable.

Category	Method example	count#	apps#	Comp%	Lib %	Indet %
Tracking_ID	UUID: randomUUID()	34395	3031	10	52	38
User_Info	GoogleSignInAccount: getAccount()	294	154	32	42	23
File_Read	File: getCanonicalFile()	1256	961	15	75	10
Hardware_ID	WifiInfo: getMacAddress()	1018	277	28	46	26
Wireless_ID	WifiInfo: getSSID()	399	113	40	49	11
Database_Read	Cursor: getString(int)	43551	4392	35	49	16
Package_Manager	PackageManager: getInstalledPackages()	1677	725	25	56	19
Media_Info	AudioRecord: read(byte[],int,int)	112	31	71	23	6
Network_Connection	URLConnection: getInputStream()	13015	3156	15	48	37
Location	Locale: getCountry()	5279	1612	35	48	17

Table 6. Sources of sensitive flows. Comp = Component, Lib = Library, Indet = Indetermined. The Method example column shows a truncated signature: Class: method-name(params)

Category	Example methods	count#	apps#	Comp%	Lib %	Indet%
Database_Write	Cursor: update(params)	5198	689	57	30	13
Log	Log:int d(params)	39449	2918	21	47	32
Shared_Preferences	SharedPreferences: putString(params)	15210	2459	21	35	44
Bundle	Bundle:void putParcelable(params)	4041	1700	23	70	7
File_Write	OutputStream:void write(byte[])	23116	3218	15	69	16
Code_Execution	Runtime: exec(java.lang.String)	19	6	17	33	50
Network_Traffic	HttpURLConnection: getOutputStream()	6359	1453	15	73	12
Media_Out	MediaRecorder: start()	8	4	100	0	0
OS_Messages	Handler:boolean sendMessage(android.os.Message)	3620	768	32	35	33
Broadcast	Context:void sendBroadcast(android.content.Intent)	1509	326	58	25	17

Table 7. Sinks of sensitive flows. Comp = Component, Lib = Library, Indet = Indeterminate. The Method example column shows a truncated signature: Class: method-name(params)

Although getting the precise geolocation is not always possible for TV apps due to hardware limitations, we detected 5279 sensitive flows in 1028 APKs (25%) corresponding to the category `Location`. The most popular information requested is country, timezone, and coarse location, usually employed to customize content. Around 35% of the APKs write location data to the Log or store it in Shared_Preferences.

We detected 31 APKs potentially exposing media information with static analysis. From this group, 12% relates to TV programs IDs, 38% other media metadata, and 41% audio recordings from hardware input. 70% of the cases are in the context of app components, but we also detected data flows that end in libraries such as Facebook and Flurry.

We detected streaming libraries not listed in any repository using the static analysis results. For instance, Penthera is a library that facilitates the delivery of video content for mobile apps, and Vizbee is a library that allows a mobile app to stream video to a TV via a native app. Vizbee's website mentions that they collect installed applications and constantly scan networks. They claim the possession of a graph of available devices across millions of homes, in which they can distinguish between private homes and public places [59]. Another library for mobile and TV apps is Nielsen SDK, which enables measuring live and on-demand TV viewing be-

havior. We found this library in the Hulu and Player TV apps. We detected data flows reading sensitive data in all cases.

5.2.2 Dynamic Analysis

We captured traffic of 21 TV apps and 22 mobile apps out of 30 `Popular-Streaming` apps. In the unsuccessful cases, we failed to collect traffic because the instrumentation broke the app, the certificate pinning modification was not successful, or geographical limitations. Table 8 summarizes the results of the traffic analysis. Figure 5 shows the top domain names collecting sensitive data in Android TV. Although some domains correspond to CDNs, most of the domains correspond to well-known trackers [14, 49, 57]. Finally, we present the comparison between mobile and TV apps in section 6.

	example	1st	Trackers	CDN
Hardware	build fingerprint	11	15	10
Location	latitude, longitude	8	3	3
Wireless	SSID, provider	3	0	0
Static Ids	android ID	5	5	0
TV metadata	Program title	11	5	3

Table 8. Traffic analysis. Number of TV apps sending sensitive data to 1st party domains, trackers, and CDNs

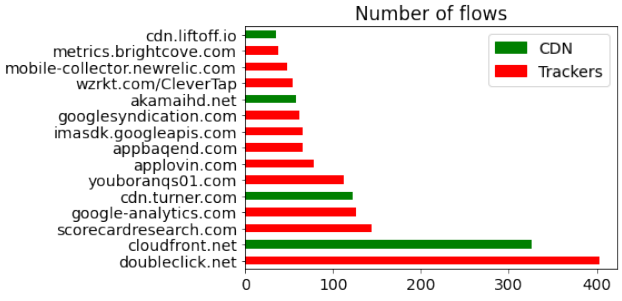

Fig. 5. Top domain names collecting sensitive data.

Identifiers. We searched static identifiers in the network traces and found that 42% of the explored TV apps (9/21) are using static identifiers. In all but one case, identifiers are sent to third-party tracking domains such as `facebook.com`, `doubleclick.net`, and `yandex.net`. In particular, we detected two groups using the same static value as advertising ID (CBNTV, RedBull, Hopper and AccuWeather as our first group, and Radio.UK, YuppTV, HTB and CNN as the second one). We also observed that the `Mobile-streaming` app `com.aetn.lifetime.watch` collects Package Manager information and the Android ID.

User Private Data. We observed 10 apps collecting geolocation data. For instance, the TV app `ru.ntv.client.tv` shares the location with `doubleclick.net` and `yandex.ru` domains. We discover that many apps (e.g., `com.cnn.mobile.android.tv`) use a third-party service to collect geolocation data based on the IP address. Although this is a legitimate service, it can be used as a side-channel to circumvent the Android permission model and user consent (apps do not declare the location permission).

We detected media metadata in the network traces of 13 TV apps (65%). It is expected to find this type of information (e.g. program title) in traffic to the app's domain and CDNs that provide multimedia content. However, several TV apps e.g., TedTalk, WWE, YuppTV, AcornTV and CNN share this information with well-known trackers such as Doubleclick, Conviva, and CleverTap. We observed the full title leaked in many cases, but we also found modified titles, which makes automated analysis more difficult.

Static VS Dynamic Analysis. The results from both analyses complement each other. Considering the `Streaming-Popular` category (65 APKs), the static analysis detected sensitive flows in 46 APKs, while the dynamic analysis captured network traces in 43. The numbers for each source category varies greatly. For instance, the dynamic analysis was more effective at detecting TV metadata flows (13 vs 2 APKs). In contrast, the static analysis found more flows (14 vs 10 APKs) for the location category. Overall, we use the static and dynamic analysis result to flag apps that we consider inappropriate. For instance, static analysis results shows evidence of potential data leaks involving multiple tracking libraries for the app `com.client.sov.adventure`. After manual inspection, we also note that this app collects network information. We confirmed that this app leaks device identifiers and geolocation to third-party domains with dynamic analysis, and it shows excessive traffic from 14 tracking domains. None of these behaviors are described in their privacy policy. We reported this app in the **Report Inappropriate Apps** section of the Play Store.

5.3 Difference Between Apps Categories

We detected 2910 sensitive data flows in 90 `Mobile-Streaming` apps, and 2190 flows in 100 `TV-streaming` apps. Both groups share 75% of the source methods. Information collected only by `Mobile-streaming` apps includes the list of installed packages and Telephony-Service methods. Telephony-Services classes are not available on Smart TVs. However, the absence of installed packages sources suggests that streaming TV apps are not accessing to this information. In contrast, we found `TV-enabled` apps exposing this information. Unique sources in `TV-Streaming` apps include Time-Zones and TV metadata from the Leanback library, which is exclusively for Android TV. Time-zones are likely used to customize streaming content.

We note similarities between the apps from the `Mobile-Streaming` and `TV-enabled` categories. However, the TV version of the popular apps present notable differences. First, we detected few TV apps with sensitive flows, only 14 out of 25 TV apps (56%) produced sensitive flows, while the other two groups are close to 90%. Second, TV apps contain considerably less sensitive data flows per APKs in all but one case. For instance, the Netflix analysis result in 10x more sensitive flows in the mobile version, and the Amazon Video in 3x.

We also looked at the difference between the network traces from the dynamic analysis. We found that many apps contact different domains for the same service (e.g., media content and tracking) on different platforms. For instance, Rakuten mobile uses Akamai as streaming provider, while the TV version uses the CenturyLink CDN. Ads domains for live streams contacted only by TV apps are `bitmovin.com` and `ooyala.com`. In contrast, `yahoo.ads`, and `platform.twitter` are domains contacted only in the mobile versions. Mobile apps like YuppTV and WWE contact social networks domains such as `facebook.com` and `twitter.com` only in the mobile version. Another example is the tracker domain `scorecardresearch.com` which is used by the CNN app in both versions. However, we observed traffic to this tracker only in the mobile version of many streaming apps. Finally, Table 9 shows a summary of the domains contacted by both version and the difference in number of flows and number of apps.

5.4 Inter-Device Communication

Smart TVs are usually in a rich environment where they can interact with other devices. In this section, we evaluate inter-device communication capabilities of TV apps. We focus our analysis on Sockets that can open remote or local connections and high-level APIs for nearby communications.

First, we manually collected all relevant methods to send/receive messages using the following APIs: Socket, NearbyConnection, NearbyMessage, and WifiDirect. Then, we search cross-references of such methods and classify the context where the methods are called. Note that our goal is to understand the potential communication capabilities of TV apps and not to measure these connections.

Table 10 shows the summary of all communication methods found in the dataset. The preferred communication method are sockets by a large margin. While we found few cases of the NearbyConnection and NearbyMessage API, we did not find any occurrence of the WifiDirect API.

Socket Communication. Overall, we detected 2646 APKs (56%) including Socket APIs. Around 90% of the APKs contain Socket methods in third-party libraries, while the remaining 10% is split between components and undetermined contexts. Specifically, we detected 382 APKs calling socket methods within apps components and a similar number for undetermined contexts. This result indicates that most of the calls to socket methods are made by third-party libraries. In fact, around 87% of the APKs contain Sockets APIs only in libraries.

Regarding to app types, the `Mobile-streaming` and `TV-streaming` categories show high occurrence of Socket calls with 92 and 95 % respectively. The number is lower for `TV-only` apps (70%) and decreases even more for `TV-enable` (51%). Previous research [34, 69] reported the number of mobile apps with open ports to be between 6 and 15% using static analysis. In contrast, we found that around half of TV apps contain socket communications.

Table 11 shows the top third-party libraries that use Sockets. We were able to identify the libraries corresponding to 90% of the socket calls using the list of third-party libraries (Section 4.2) and manual analysis. The library with the most matches is Okio, a library that facilitates dealing file access and HTTP requests.

Figure 6 shows an overview of socket usage classified by library category. One could argue that developers are aware of sockets for some categories like communication

domain	Description	#TV flows	#TV apps	#Mobile flows	#Mobile apps
akamai	CDN/streaming	573	6	537	7
doubleclick	ads/tracking	44	4	159	9
google-analytics	tracking	51	3	98	6
facebook	ads/tracking	9	1	144	7
scordcardresearch	tracking	36	1	38	4
appsflyer	ads/tracking	5	1	41	2

Table 9. Top domains contacted by TV and mobile streaming apps

API	methods	Total APKs	Comp #	Lib #	Both #	Ind #	Total calls
Sockets	send_data	2644	362	2408	249	574	8378
	receive_data	2451	176	2246	113	472	4384
NearbyConnection	send_message	5	5	0	0	0	8
	receive_message	8	8	0	0	0	32
	discovery/advertise	5	5	0	0	0	10
NearbyMessage	publish	3	3	0	0	0	3
	subscribe	3	3	0	0	0	3

Table 10. Summary of communication APIs. Comp = Component, Lib = Library, Indet = Indeterminate, Both = Lib and Comp. These results are independent of the ones in section 5.1

Library	Description	# apps
Okio	I/O, networking	1540
Mintegral	Ads-Analytics	233
Apache	Utilities	227
Facebook	Ads-Analytics	218
Yandex	Ads-Analytics	178
Fyber	Ads-Analytics	169

Table 11. Top libraries using sockets

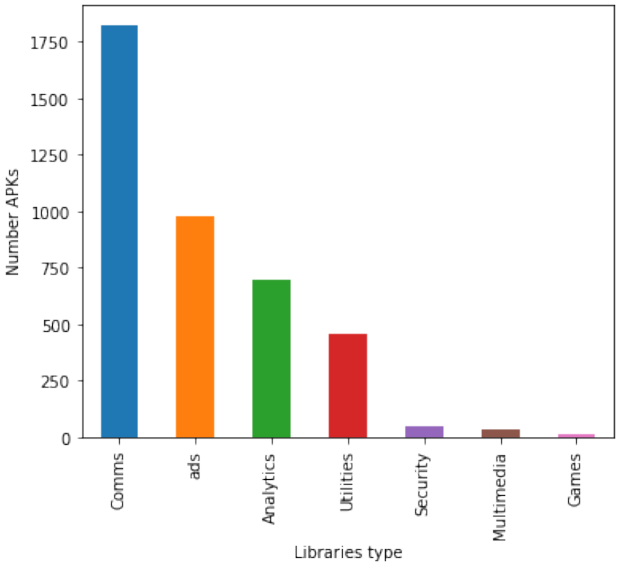

Fig. 6. Libraries using sockets grouped by category

and security (such as the case of Okio). However, it is unclear how informed are the developers of other categories such as advertising and tracking.

Nearby Communications. We found only 12 APKs using the `NearbyConnection` and `NearbyMessages` APIs. Notably, the difference with sockets is that all method calls are made within the app's components. Google guidelines' state that connections established without authentication are insecure [27], so we also check whether the apps are implementing authentication by searching calls to the methods that read the authentication token in the callback `onConnectionInitiated`. Unfortunately, none of the APKs that use the `NearbyConnection` API implement authentication. Although this step is optional, a connection without authentication opens an insecure channel that could be exploited [6].

The Nearby platform APIs are a good alternative to implement two-way communication without requiring an Internet connection, but its adoption is very low in our dataset. Meftah *et al.* already reported this trend in the Play Store [42]. They argue that constant crashes, battery issues, and testing difficulties contribute to the lack of adoption, which is exacerbated by bad user reviews. Overall, TV apps developers still rely on plain sockets to communicate with other devices.

Old VS New APIs. Many apps we analyzed include inter-device communication features. Our work shows

that developers prefer to implement sockets rather than the new APIs from the Nearby platform, and WiFi Direct. These libraries have been developed since 2014 and have built-in encryption. In contrast, implementing secure communication over a socket is normally more complex and prone to bugs. The fact that developers are not taking up efforts made by the manufacturer to develop more easy-to-use and secure libraries shows flaws in the approach. What is worse, we showed that the few developers that use the newer libraries, do not use them properly. We have notified the apps developers of this issue. Last, we detected many socket calls in communication libraries, which suggests that the intention is well-known, but there are also instances of tracking libraries including socket APIs. Although TV apps use sockets predominantly in TPL, the number of socket calls in the app's components vastly exceeds the use of newer APIs.

6 Discussions

In this section, we reflect on our findings, how they portray the Android TV ecosystem, and how they are related to the mobile ecosystem. We also discuss the prevalence of malware in our dataset.

6.1 TV Apps VS Mobile Apps

Our results show that mobile apps request on average 20% more normal and dangerous AOSP permissions than TV apps. This is in line with the additional resources requiring permissions in the mobile ecosystem that are not available in Android TV. In fact, we found some permissions that are unique to each platform. Permissions related to sensors, orientation, and user dictionaries are found only in mobile apps (and heavily requested). In contrast, TV apps request permissions for capturing TV inputs, content rating system, and HDMI settings.

TV apps request 15% more custom permissions than mobile apps. This difference indicates more third-party dependencies in Android TV. Since Android TV and Android mobile are highly compatible, TPLs are reused making the TPLs similar on both platforms. Nevertheless, some libraries are unique to Android TV, generally for codec management and TV-related hardware. From the mobile perspective, we found libraries that facilitate the interaction with Smart TVs showing abusive data collection practices. Regarding sensitive data, the most noticeable difference is that mobile apps rely on precise geolocation. In contrast, TV apps request coarse location and with less frequency.

We see the different categories of TV apps having similar behaviors, but mobile and streaming TV apps show a higher occurrence of socket APIs than other categories. TV apps differ from mobile apps in that TV apps are updated less frequently. Most of the e `Mobile_streaming` apps (94%) were updated in 2021, while for `TV_only` this number was 75%.

Popular Streaming Apps. By analyzing the Play Store metadata we noted a significant degradation in the quality of the TV version of mobile apps. This degradation is evident when looking at users' rating, reviews, and last app update from developers. For instance, all but one TV app have a lower user rating than the mobile version. Significantly, 55% have at least 2 points of difference on a scale of 1 to 5 stars. Two examples of this are the YouTube apps (mobile version 4.3 vs TV 2.2), and CNN apps (mobile 4.5 vs TV 1.3). This difference is more evident when reading the reviews on the Play Store where users constantly complain about buggy and unusable apps. We also note that popular streaming apps perform more regular updates of their TV_only apps with a few months delay compared to their mobile counterpart. Nevertheless, there are two TV apps that were left without updates since 2018 (`com.playstation.video.atv` and `com.ted.android.tv`). Our TPL analysis detected 40% more libraries in mobile apps, and more matches per library. For instance, the number of mobile apps with the Facebook SDK doubles that of TV apps.

6.2 Data Collection Practices

Our results show that TV apps rely heavily on static identifiers. Google recommends the class `AdvertisingIdInfo` for tracking, as it allows users to easily reset the value [29]. The fact that we found few resettable identifiers means that policies of restricting advertising IDs [22] will have no real impact in the Smart TV ecosystem. In contrast, we found a prevalence of static identifiers, such as GUIDs, and hardware addresses.

Libraries are responsible for around 60% of all sensitive flows transporting identifiers. These flows are present in 85% of the TV apps. In our analysis, we detected that around 90% of the TV apps include advertisement or analytics libraries, some of them with a bad

history of exposing sensitive data such as Umeng, Appodeal, and Unity. What is worse, it is well known that many developers are not aware of the data collection practices by TPL, which poses an even greater threat to end users [12, 37, 52].

Our dynamic analysis experiments confirmed that many TV apps use static identifiers to track users, and that these IDs are shared with tracker domains. We detected static identifiers in the network traces by comparing IDs found across different apps. However, our analysis cannot identify all dynamic IDs generated by different tracking SDKs.

A noteworthy example is the TV companion app `xyz.klinker.messenger` that sends texts/multimedia messages. This app claims to be completely free, with no ads and no personal data collection in its Play Store description. However, it requests permissions to access the geolocation, user profile, and custom permissions from vendors like Huawei and Sony. Moreover, the app's privacy policy mentions collecting personal data (in contradiction with its description) such as social media, geolocation, OS information, and the list of third-party providers. We also noted that this app requests all dangerous permissions immediately after launch, which is against the best practices [24] because it obscures the association of run-time permissions with a specific action. We also reported this app to Google.

6.3 Malicious Behaviors in Android TV

We detected 34 APKs in our dataset that are flagged as malware by more than 10 engines in VirusTotal (VT). 6 of these APKs are from the `TV_only` category, of which one was removed from the Play Store, three are paid apps, and two are VPNs. Notably, the APKs with higher VT score (25-34) are games. These APKs include payloads capable of remote execution, capture inputs, sensitive data collection, fraudulent ads, and silently install packages according to the VirusTotal evaluation. In many cases, these apps rely on normal permission to execute malicious behavior, for instance, to display a window over other apps, install packages or read the WiFi state.

We attribute the APKs with malware to 22 developers using their certificates. Based on this, we were able to identify an additional set of 44 APKs belonging to these developers (10 `TV_only`). Most of the added APKs have a low VT score. However, one noteworthy case is the `TV China` app that offer IPTV services. We found this app under two different package names (`com.live.tv.home`, and `com.cntv.movies`). The VT score are 7 and 9 respectively, and their evaluation is almost identical, with 7 months of difference between the evaluations. Both apps were removed from the Play Store by the time of writing of this paper.

A significant factor limiting user control over these threats is that the Play Store UI in Android TV does not provide the same information as the mobile version. There is no way for the user to check permissions, read reviews, or the privacy policy of the app. While users can grant or revoke dangerous permissions, they do not have control over normal permissions. In view of the evidence of malicious behavior and permission misuse, we argue that the Android TV app store should include additional information on their app pages so users can take more informed decisions (at least to the same level as mobile users). Additionally, better TV apps guidelines for developers can help with problems such as overprivileged and inconsistent apps.

7 Limitations

Ecosystem. Our dataset is limited to around 4.5k unique packages. This is larger than previous studies of Smart TVs [45, 57]. However, the heuristic used to search TV apps might have missed part of the ecosystem. Particularly, TV apps from underrepresented regions could be included by searching in other markets. Self-signed certificates provide limited guarantees about a developer information, but it is still the most reliable approach to identify developers at a large scale [20]. Alternative approaches rely on fingerprinting and Machine Learning [35, 64]. These techniques can complement our approach based on certificate fingerprint and custom permission.

Permissions. We only consider static definitions when analyzing custom permissions. Omitting dynamic definitions can generate incomplete results. However, a previous study showed that only 3% of custom permission are declared dynamically [55], making our results representative. The permission rationale analysis conservatively assumes that an app shows the rationale if the context of relevant API calls matches. A more precise analysis could search data dependencies between the methods or use dynamic analysis.

Third-Party Libraries The accuracy of the TPL detection using Libscout depends on the completeness of the library profiles and the limitations of Libscout such as package flattening and dead code elimination. In con-

trast to Libscout, our second approach is not resilient to obfuscated package names, but still our implementation allow us to detect many common libraries.

Behavioral Analysis. Static analysis abstracts from user inputs and approximate runtime values which can be a source of false positives. Additionally, our static analysis does not consider dynamic code loading, some cases of reflection, and native code. We detected that 6% the apps in our dataset load dynamic code, while 60% include the required classes to do it. While we use dynamic analysis to overcome these limitations, our traffic analysis experiment does not scale, mainly because we decided to log in on each app during testing to get more accurate results and the time limit of 15 minutes.

8 Related Work

There are several works that have studied different aspects of the mobile ecosystem. Wang *et al.* presented a large-scale analysis of the Play Store, where they discovered a 25% increase of ad libraries over a period of three years and the decrease of paid apps [61]. Similarly, other studies analyzed Chinese markets [63], mobile ecosystem in general including TPL [12, 14, 40, 49, 71], tracking and advertising ecosystem [14, 37, 43, 49, 63], and trust in the Android ecosystem [20, 62, 64]. Although similar, the Android TV ecosystem has its peculiarities and has not been the focus of previous research.

Several studies have used static analysis to detect potential information leaks and harmful behavior in Android apps [8, 15, 30, 66]. Due to their popularity as a communication method, there have been previous research efforts looking at sockets to detect vulnerabilities and data leakage [16, 34, 69]. We consider both approaches to analyze potential information leaks and vulnerable communication in TV apps.

Previous works reported vulnerabilities and data leakage in IoT devices, including Smart TVs and Chromecast devices [33, 36, 46, 51, 54]. Recently, two papers investigated security and privacy issues on Roku and Fire TV using traffic analysis [45, 57]. They found that a large number of TV apps expose PII to third-party domains. They reported mixed results of DNS-based blocklist to prevent data leakage, with concerning results on static identifier leaks. In contrast, we use static analysis which allow us to increase the size of the dataset and detect information flows carry on static identifiers and other sensitive data.

Finally, other studies in this area have also analyzed several attacks on Smart TVs [11, 44], review of privacy and security aspects [2], and user expectations regarding privacy [41]. Our paper complements these works by analyzing several issues regarding the certificate of TV apps and permission misuse such as inconsistent and over-privileged apps which are problems reported in the mobile ecosystem [38, 55, 70, 72].

9 Conclusion

In this work, we have presented the first systematic study of the Android TV ecosystem. The analysis of more than 4.5k TV apps reveals pervasive sensitive data collection and tracking practices. Notably, we detected the prevalence of static identifiers over identifiers designed to protect users' privacy. Consequently, policies such as limiting access to advertising identifiers will have almost no effect on TV apps. Our measurements show that tracking and advertising services collect static identifiers and personal data in a large percentage of the cases, including TV specific data.

While our static and dynamic analyses show that TV apps have consistently fewer trackers than mobile apps, our work also sheds light on the developer ecosystem detecting multiple bad development practices that leave TV customers vulnerable. Moreover, developers porting mobile apps to Android TV produced inconsistent apps with much lower quality, including popular streaming services. We expect that our study contributes to the Android TV ecosystem to improve development practices and general design. In particular, we encourage the development of better guidelines to migrate mobile apps to Android TV and secure usage of new APIs for inter-device communication. Finally, we feel that Android TV should improve its UI, allowing users to at least check permissions and read privacy policies before installing apps on their TVs.

Acknowledgements

This research has been partially sponsored by the Engineering and Physical Sciences Research Council (EPSRC) and the UK government as part of the Centre for Doctoral Training (CDT) in Cyber Security for the Everyday at Royal Holloway, University of London (EP/P009301/1).

References

[1] Gunes Acar, Danny Yuxing Huang, Frank Li, Arvind Narayanan, and Nick Feamster. Web-based attacks to discover and control local iot devices. In *Proceedings of the 2018 Workshop on IoT Security and Privacy*, pages 29–35, 2018.

[2] Iftikhar Alam, Shah Khusro, and Muhammad Naeem. A review of smart tv: Past, present, and future. In *2017 International Conference on Open Source Systems & Technologies (ICOSST)*, pages 35–41. IEEE, 2017.

[3] Moutaz Alazab, Mamoun Alazab, Andrii Shalaginov, Abdelwadood Mesleh, and Albara Awajan. Intelligent mobile malware detection using permission requests and api calls. *Future Generation Computer Systems*, 107:509–521, 2020.

[4] Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. Androzoo: Collecting millions of android apps for the research community. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 468–471, New York, NY, USA, 2016. ACM.

[5] Julio Angulo, Simone Fischer-Hübner, Tobias Pulls, and Erik Wästlund. Usable transparency with the data track: a tool for visualizing data disclosures. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems*, pages 1803–1808, 2015.

[6] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. Nearby threats: Reversing, analyzing, and attacking google's' nearby connections' on android. In *Network and Distributed System Security Symposium*, 2019.

[7] APKMirror. Apkmirror website. Accessed March 2021. https://www.apkmirror.com/.

[8] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *Acm Sigplan Notices*, 49(6):259–269, 2014.

[9] Parks Associates. Future of video. Accessed March 2021. http://www.parksassociates.com/events/future-of-video/fov-2018-pr5.

[10] Kathy Wain Yee Au, Yi Fan Zhou, Zhen Huang, and David Lie. Pscout: analyzing the android permission specification. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 217–228, 2012.

[11] Yann Bachy, Frédéric Basse, Vincent Nicomette, Eric Alata, Mohamed Kaâniche, Jean-Christophe Courrege, and Pierre Lukjanenko. Smart-tv security analysis: practical experiments. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 497–504. IEEE, 2015.

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

[13] Venture Beat. Android ads triggered up to 21%. Accessed March 2021. https://venturebeat.com/2021/05/19/post-idfa-alliance-finds-ios-14-5-triggered-up-to-21-growth-in-android-ad-spending/.

[14] Reuben Binns, Ulrik Lyngs, Max Van Kleek, Jun Zhao, Timothy Libert, and Nigel Shadbolt. Third party tracking in the mobile ecosystem. In *Proceedings of the 10th ACM Conference on Web Science*, pages 23–31, 2018.

[15] Amiangshu Bosu, Fang Liu, Danfeng Yao, and Gang Wang. Collusive data leak and more: Large-scale threat analysis of inter-app communications. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 71–85, 2017.

[16] Wenqi Bu, Minhui Xue, Lihua Xu, Yajin Zhou, Zhushou Tang, and Tao Xie. When program analysis meets mobile security: an industrial study of misusing android internet sockets. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pages 842–847, 2017.

[17] Federal Trade Commission. Vizio to pay $2.2 million to ftc. Accessed March 2021. https://www.ftc.gov/news-events/press-releases/2017/02/vizio-pay-22-million-ftc-state-new-jersey-settle-charges-it.

[18] Tech Crunch. Google removes apps for children. Accessed March 2021. https://techcrunch.com/2020/10/23/google-removes-3-android-apps-for-children-with-20m-downloads-between-them-over-data-collection-violations/.

[19] Yusra Elbitar, Michael Schilling, Trung Tin Nguyen, Michael Backes, and Sven Bugiel. Explanation beats context: The effect of timing & rationales on users' runtime permission decisions. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 785–802. USENIX Association, August 2021.

[20] Julien Gamba, Mohammed Rashed, Abbas Razaghpanah, Juan Tapiador, and Narseo Vallina-Rodriguez. An analysis of pre-installed android software. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1039–1055. IEEE, 2020.

[21] Google. Androguard documentation. Accessed March 2021. https://github.com/androguard/androguard.

[22] Google. Android advertising id policy change. Accessed March 2021. https://support.google.com/googleplay/android-developer/answer/6048248.

[23] Google. Android blog support. Accessed March 2021. https://support.google.com/googleplay/android-developer/answer/10158779?hl=en.

[24] Google. Android permissions notes. Accessed March 2021. https://developer.android.com/training/permissions/usage-note.

[25] Google. Android tv guide. Accessed March 2021. https://developer.android.com/training/tv/start/start.

[26] Google. Google permissions guide. Accessed March 2021. https://developer.android.com/guide/topics/permissions/defining.

[27] Google. Nearby connections guide. Accessed March 2021. https:https://developers.google.com/nearby/connections/android/manage-connections#authenticate_a_connection.

[28] Google. Privacy security best practices. Accessed March 2021. https://source.android.com/security/best-practices/privacy.

[29] Google. Unique identifiers guide. Accessed March 2021. https://developer.android.com/training/articles/user-data-ids.

[30] Michael I Gordon, Deokhwan Kim, Jeff H Perkins, Limei Gilham, Nguyen Nguyen, and Martin C Rinard. Information flow analysis of android applications in droidsafe. In *NDSS*,

volume 15, page 110, 2015.

[31] Privacy Grade. Privacy grade website. Accessed March 2021. http://privacygrade.org/.

[32] Niklas Higi. apk-mitm repository. Accessed December 2021. https://github.com/shroudedcode/apk-mitm.

[33] Danny Yuxing Huang, Noah Apthorpe, Frank Li, Gunes Acar, and Nick Feamster. Iot inspector: Crowdsourcing labeled network traffic from smart home devices at scale. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 4(2):1–21, 2020.

[34] Yunhan Jack Jia, Qi Alfred Chen, Yikai Lin, Chao Kong, and Z Morley Mao. Open doors for bob and mallory: Open port usage in android apps and security implications. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 190–203. IEEE, 2017.

[35] Vaibhavi Kalgutkar, Natalia Stakhanova, Paul Cook, and Alina Matyukhina. Android authorship attribution through string analysis. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, pages 1–10, 2018.

[36] Deepak Kumar, Kelly Shen, Benton Case, Deepali Garg, Galina Alperovich, Dmitry Kuznetsov, Rajarshi Gupta, and Zakir Durumeric. All things considered: an analysis of iot devices on home networks. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 1169–1185, 2019.

[37] Li Li, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. An investigation into the use of common libraries in android apps. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 1, pages 403–414. IEEE, 2016.

[38] Rui Li, Wenrui Diao, Zhou Li, Jianqi Du, and Shanqing Guo. Android custom permissions demystified: From privilege escalation to design shortcomings. *2021 IEEE Symposium on Security and Privacy (SP)*, 2021.

[39] Libscout. Library profiles, libscout. Accessed March 2021. https://github.com/reddr/LibScout-Profiles.

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

[41] Nathan Malkin, Julia Bernd, Maritza Johnson, and Serge Egelman. "what can't data be used for?" privacy expectations about smart tvs in the us. In *Proceedings of the 3rd European Workshop on Usable Security (EuroUSEC), London, UK*, 2018.

[42] Lakhdar Meftah, Romain Rouvoy, and Isabelle Chrisment. Testing nearby peer-to-peer mobile apps at large. In *2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, pages 1–11. IEEE, 2019.

[43] Georg Merzdovnik, Markus Huber, Damjan Buhov, Nick Nikiforakis, Sebastian Neuner, Martin Schmiedecker, and Edgar Weippl. Block me if you can: A large-scale study of tracker-blocking tools. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 319–333. IEEE, 2017.

[44] Benjamin Michéle and Andrew Karpow. Watch and be watched: Compromising all smart tv generations. In *2014 IEEE 11th Consumer Communications and Networking Conference (CCNC)*, pages 351–356. IEEE, 2014.

[45] Hooman Mohajeri Moghaddam, Gunes Acar, Ben Burgess, Arunesh Mathur, Danny Yuxing Huang, Nick Feamster, Edward W Felten, Prateek Mittal, and Arvind Narayanan. Watching you watch: The tracking ecosystem of over-the-top tv streaming devices. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 131–147, 2019.

[46] Alexios Nikas, Efthimios Alepis, and Constantinos Patsakis. I know what you streamed last night: On the security and privacy of streaming. *Digital Investigation*, 25:78–89, 2018.

[47] Fabio Pierazzi, Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo Cavallaro. Intriguing properties of adversarial ml attacks in the problem space. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1332–1349. IEEE, 2020.

[48] Lina Qiu, Yingying Wang, and Julia Rubin. Analyzing the analyzers: Flowdroid/iccta, amandroid, and droidsafe. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 176–186, 2018.

[49] Abbas Razaghpanah, Rishab Nithyanand, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Mark Allman, Christian Kreibich, and Phillipa Gill. Apps, trackers, privacy, and regulators: A global study of the mobile tracking ecosystem. In *Network and Distributed System Security Symposium*, 2018.

[50] Joel Reardon, Álvaro Feal, Primal Wijesekera, Amit Elazari Bar On, Narseo Vallina-Rodriguez, and Serge Egelman. 50 ways to leak your data: An exploration of apps' circumvention of the android permissions system. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 603–620, 2019.

[51] Jingjing Ren, Daniel J Dubois, David Choffnes, Anna Maria Mandalari, Roman Kolcun, and Hamed Haddadi. Information exposure from consumer iot devices: A multidimensional, network-informed measurement approach. In *Proceedings of the Internet Measurement Conference*, pages 267–279, 2019.

[52] Gian Luca Scoccia, Ibrahim Kanj, Ivano Malavolta, and Kaveh Razavi. Leave my apps alone! a study on how android developers access installed apps on user's device. In *Proceedings of the IEEE/ACM 7th International Conference on Mobile Software Engineering and Systems*, pages 38–49, 2020.

[53] Silvia Sebastian and Juan Caballero. Towards attribution in mobile markets: identifying developer account polymorphism. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 771–785, 2020.

[54] Vijay Sivaraman, Hassan Habibi Gharakheili, Clinton Fernandes, Narelle Clark, and Tanya Karliychuk. Smart iot devices in the home: Security and privacy implications. *IEEE Technology and Society Magazine*, 37(2):71–79, 2018.

[55] Güliz Seray Tuncay, Soteris Demetriou, Karan Ganju, and C Gunter. Resolving the predicament of android custom permissions. In *Network and Distributed System Security Symposium*. Internet Society, 2018.

[56] Max Van Kleek, Ilaria Liccardi, Reuben Binns, Jun Zhao, Daniel J Weitzner, and Nigel Shadbolt. Better the devil you

know: Exposing the data sharing practices of smartphone apps. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 5208–5220, 2017.

[57] Janus Varmarken, Hieu Le, Anastasia Shuba, Athina Markopoulou, and Zubair Shafiq. The tv is smart and full of trackers: Measuring smart tv advertising and tracking. *Proceedings on Privacy Enhancing Technologies*, 2020(2):129–154, 2020.

[58] The Verge. Fertility app shared customer data with chinese companies. Accessed March 2021. https://www.theverge.com/2020/8/20/21377591/fertility-app-premom-reportedly-shared-customer-data-with-chinese-companies.

[59] Vizbee. Vizbee developers overview. Accessed March 2021. https://vzb.us/overview/.

[60] Karl von Randow. Charles proxy homepage. Accessed December 2021. https://www.charlesproxy.com/.

[61] Haoyu Wang, Hao Li, and Yao Guo. Understanding the evolution of mobile app ecosystems: A longitudinal measurement study of google play. In *The World Wide Web Conference*, pages 1988–1999, 2019.

[62] Haoyu Wang, Hongxuan Liu, Xusheng Xiao, Guozhu Meng, and Yao Guo. Characterizing android app signing issues. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 280–292. IEEE, 2019.

[63] Haoyu Wang, Zhe Liu, Jingyue Liang, Narseo Vallina-Rodriguez, Yao Guo, Li Li, Juan Tapiador, Jingcun Cao, and Guoai Xu. Beyond google play: A large-scale comparative study of chinese android app markets. In *Proceedings of the Internet Measurement Conference 2018*, pages 293–307, 2018.

[64] Wei Wang, Guozhu Meng, Haoyu Wang, Kai Chen, Weimin Ge, and Xiaohong Li. A 3 ident: A two-phased approach to identify the leading authors of android apps. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 617–628. IEEE, 2020.

[65] Fengguo Wei, Yuping Li, Sankardas Roy, Xinming Ou, and Wu Zhou. Deep ground truth analysis of current android malware. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 252–276. Springer, 2017.

[66] Fengguo Wei, Sankardas Roy, and Xinming Ou. Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 1329–1341, 2014.

[67] Business Wire. Smart tv market share. Accessed March 2021. https://www.businesswire.com/news/home/20210521005413/en/193.23-Billion-Global-Smart-TV-Market-COVID-19-Growth-and-Change-to-2030---ResearchAndMarkets.com.

[68] WIRED. Millions of google, roku, and sonos devices are vulnerable to a web attack | wired. Accessed March 2021. https://www.wired.com/story/chromecast-roku-sonos-dns-rebinding-vulnerability/..

[69] Daoyuan Wu, Debin Gao, Rocky KC Chang, En He, Eric KT Cheng, and Robert H Deng. Understanding open ports in android applications: Discovery, diagnosis, and security assessment. In *Network and Distributed System Security Symposium*. Internet Society, 2019.

[70] Sha Wu and Jiajia Liu. Overprivileged permission detection for android applications. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2019.

[71] Yuan Zhang, Jiarun Dai, Xiaohan Zhang, Sirong Huang, Zhemin Yang, Min Yang, and Hao Chen. Detecting third-party libraries in android applications with high precision and recall. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 141–152. IEEE, 2018.

[72] Yuan Zhang, Min Yang, Bingquan Xu, Zhemin Yang, Guofei Gu, Peng Ning, X Sean Wang, and Binyu Zang. Vetting undesirable behaviors in android apps with permission use analysis. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 611–622, 2013.

[73] Xiaoyong Zhou, Soteris Demetriou, Dongjing He, Muhammad Naveed, Xiaorui Pan, XiaoFeng Wang, Carl A Gunter, and Klara Nahrstedt. Identity, location, disease and more: Inferring your secrets from android public resources. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1017–1028, 2013.

A Dataset Collection

We collect our app dataset with two crawlers from two main sources: Google Play Store (via AndroZoo) and ApkMirror directly. AndroZoo is a repository that constantly crawls the Play Store and other markets and provide an API to download APKs at high rates. Once we download an APK, we verify if it can run on Smart TVs by checking the APK Manifest, and we discard the file if it does not meet the criteria. We also remove duplicate APKs with the same package name.

While the list of all 4745 package names can be found in the repository, we provide here the names of the popular streaming apps used for the dynamic analysis and to compare the mobile/TV version of apps:

- TV/Mobile pairs: BBCPlayer, ITV, Rakuten, CNN, AcornTV, Lifetime, Amazon Video, CBNTV, Christian Channel, Curiosity Stream, Earthcam, Eros, YouTube, Stadia, Moviestar, Hulu, Netflix, PlayStation, AccuWeather, TedTalk, YuppTV, WRC, Player, Tvn, HTB
- Unique version: Old Movies, Disney+, Twitch, Pluto, NFL, Plex, Sling, Natgeo, Espn, Apple TV, RedBull, Kodi, HayStack, NBC Sports, HBOGo