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

**Abstract:** In this paper, we present a large-scale measurement study of the smart TV advertising and tracking ecosystem. First, we illuminate the network behavior of smart TVs as used in the wild by analyzing network traffic collected from residential gateways. We find that smart TVs connect to well-known and platform-specific advertising and tracking services (ATSes). Second, we design and implement software tools that systematically explore and collect traffic from the top-1000 apps on two popular smart TV platforms, Roku and Amazon Fire TV. We discover that a subset of apps communicate with a large number of ATSes, and that some ATS organizations only appear on certain platforms, showing a possible segmentation of the smart TV ATS ecosystem across platforms. Third, we evaluate the (in)effectiveness of DNS-based blocklists in preventing smart TVs from accessing ATSes. We highlight that even smart TV-specific blocklists suffer from missed ads and incur functionality breakage. Finally, we examine our Roku and Fire TV datasets for exposure of personally identifiable information (PII) and find that hundreds of apps exfiltrate PII to third parties and platform domains. We also find evidence that some apps send the advertising ID alongside static PII values, effectively eliminating the user's ability to opt out of ad personalization.

**Keywords:** Smart TV; privacy; tracking; advertising; blocklists

DOI 10.2478/popets-2020-0021
Received 2019-08-31; revised 2019-12-15; accepted 2019-12-16.

***Corresponding Author: Janus Varmarken†:** University of California, Irvine, E-mail: jvarmark@uci.edu
**Hieu Le†:** University of California, Irvine, E-mail: hieul@uci.edu. († *The first two authors made equal contributions and share the first authorship).*
**Anastasia Shuba:** Broadcom Inc. (The author was a student at the University of California, Irvine at the time the work was conducted), E-mail: ashuba@uci.edu
**Athina Markopoulou:** University of California, Irvine, E-mail: athina@uci.edu
**Zubair Shafiq:** University of Iowa, E-mail: zubair-shafiq@uiowa.edu

## 1 Introduction

Smart TV adoption has steadily grown over the last few years, with more than 37% of US households owning at least one smart TV in 2018, a 16% increase over the previous year [1]. This growth is driven by two trends. First, over-the top (OTT) video streaming services such as Hulu and Netflix have become popular, with more than 28 million and 60 million subscribers in the US, respectively [2]. Second, smart TV solutions are available at relatively affordable prices, with many of the external smart TV boxes/sticks priced less than $50, while built-in smart TVs now cost only a few hundreds dollars [3]. As a result, a diverse set of smart TV platforms have emerged, and have been integrated into various smart TV products. For example, Apple TV integrates tvOS, and TCL and Sharp TVs integrate Roku.

Many of these platforms have their own respective app store, where the vast majority of smart TV apps are ad-supported [4]. OTT advertising, which includes smart TV, is expected to increase by 40% to $2 billion in 2018 [5]. Roku and Fire TV are two of the leading smart TV platforms in number of ad requests [6]. Despite their increasing popularity, the advertising and tracking services ("ATSes") on smart TVs are currently not well understood by users, researchers, and regulators. In this paper, we present one of the first large-scale measurement studies of the emerging smart TV advertising and tracking ecosystem.

**In the Wild Measurements (§3).** First, we analyze the network traffic of smart TV devices in the wild. We instrument residential gateways of 41 homes and collect flow-level summary logs of the network traffic generated by 57 smart TVs from seven different platforms. The comparative analysis of network traffic by different smart TV platforms uncovers similarities and differences in their characteristics. As expected, we find that a substantial fraction of the traffic is related to popular video streaming services such as Netflix and Hulu. More importantly, we find generic as well as platform-specific ATSes. Although realistic, the in the wild dataset does not provide app-level visibility, *i.e.*, we cannot determine which apps generate traffic to ATSes. To address this limitation, we undertake the following major effort.

**Controlled Testbed Measurements (§4).** We design and implement two software tools, *Rokustic* for Roku and *Firetastic* for Amazon Fire TV, which systematically explore apps and collect their network traffic. We use Rokustic and Firetastic to exercise the top-1000 apps of their respective platforms, and refer to the collected network traffic as our *testbed* datasets. We analyze the testbed datasets w.r.t. the top Internet destinations the apps contact, at the granularity of fully qualified domain names (FQDNs), effective second level domains (eSLDs), and organizations. We use "domain", "endpoint", and "destination" interchangeably in place of FQDN and eSLD when the distinction is clear from the context. We further separate destinations as first, third, and platform-specific party, w.r.t. to the app that contacts them.

First, we find that the majority of apps contact few ATSes, while about 10% of the apps contact a large number of ATSes. Interestingly, many of these more concerning apps come from a small set of developers. Second, we find what appears to be a segmentation of the smart TV ATS ecosystem across Roku and Fire TV as (1) the two datasets have little overlap in terms of ATS domains; (2) some third party ATSes are among the key players on one platform, but completely absent on the other; and (3) apps that are present on both platforms have little overlap in terms of the domains they contact. Third, we compare the top third party ATS domains of the testbed datasets to those of Android.

**Evaluation of DNS-Based Blocklists (§5).** Users typically rely on DNS-based blocking solutions such as Pi-hole [7] to prevent in-home devices such as smart TVs from accessing ATSes. Thus, we evaluate the effectiveness of DNS-based blocklists, selecting those that are most relevant to smart TVs. Specifically, we examine and test four popular blocklists: (1) Pi-hole Default blocklist (PD) [7], (2) Firebog's recommended advertising and tracking lists (TF) [8], (3) Mother of all Ad-Blocking (MoaAB) [9], and (4) StopAd's smart TV specific blocklist (SATV) [10]. Our comparative analysis shows that block rates vary, with Firebog having the highest coverage across different platforms and StopAd blocking the least. We further investigate potential false negatives (FN) and false positives (FP). We discover that blocklists miss different ATSes (FN), some of which are missed by all blocklists, while more aggressive blocklists can suffer from false positives that result in breaking app functionality. We discuss two ways to discover false negatives, through observing domains contacted by multiple apps ("app prevalence") and keyword search (based on ATS related words like "ads" and "measure").

**PII Exposures (§6).** We further examine the network traces of our testbed datasets and find that hundreds of apps exfiltrate personally identifiable information (PII) to third parties and platform-specific parties, mostly for non-functional advertising and tracking purposes. Alarmingly, we find that many apps send the advertising ID alongside static PII values such as the device's serial number. This eliminates the user's ability to opt out of personalized advertisements by resetting the advertising ID, since the ATS can simply link an old advertising ID to its new value by joining on the serial number. We evaluate the blocklists' ability to prevent exposures of PII and find that they generally perform well for Roku, but struggle to prevent exfiltration of the device's serial number and the device ID to third parties and the platform-specific party for Fire TV.

**Contributions.** In this paper, we analyze the network behavior of smart TVs, both in the wild and in the lab. Our contributions include the following: (1) providing an in-depth comparative analysis of the ATS ecosystems of Roku, Fire TV, and Android; (2) illuminating the key players within the Roku and Fire TV ATS ecosystems by mapping domains to eSLDs and parent organizations; (3) evaluating the effectiveness and adverse effects of an extensive set of blocklists, including smart TV specific blocklists; (4) instrumenting long experiments per app to uncover approximately twice as many domains as [11]; and (5) making our tools, Rokustic and Firetastic, and our testbed datasets available [12].

**Outline.** The structure of the rest of the paper is as follows. Section 2 provides background on smart TVs and reviews related work. Section 3 presents the in the wild measurement and analysis of 57 smart TVs from seven different platforms in 41 homes. Section 4 presents our systematic testing approach and comparative analysis of the top-1000 Roku and Fire TV apps. Section 5 evaluates four well-known DNS-based blocklists and shows their limitations through analysis of missed ATSes and app breakage. Section 6 investigates exfiltration of PII. Section 7 concludes the paper, discusses limitations, and outlines directions for future work. The appendix provides additional details and results, as well as an evaluation and discussion of the limitations of our approach.

## 2 Background & Related Work

**Background on Smart TVs.** A smart TV is essentially an Internet-connected TV that has apps for users to stream content, play games, and even browse the web. There are two types of smart TV products in the market: (1) built-in smart TVs, and (2) external smart TV boxes/sticks, also referred to as over-the-top (OTT) streaming devices. Some TV manufacturers, such as Samsung and Sony, offer TVs with built-in smart TV functionality. While several others provide external box/stick solutions, such as Roku (by Roku), Fire TV (by Amazon), and Apple TV (by Apple), that convert a regular TV into a smart TV. In addition, hybrid products exist, as some TV manufacturers now integrate external box/stick solutions directly into their smart TVs. For example, TCL and Sharp offer smart TVs that integrate Roku TV, while Insignia and Toshiba offer smart TVs that integrate Fire TV. We use "smart TV" as an umbrella term for TVs with built-in smart TV functionality and OTT streaming devices.

There is a diverse set of smart TV platforms, each with its own set of apps that users can install on their TVs. Many smart TVs use an Android-based operating system (*e.g.*, Sony, AirTV, Philips) or a modified version of it (*e.g.*, Fire TV). Regular Android TVs have access to apps from the Google Play Store, while Fire TV has its own app store controlled by Amazon. In both cases, applications for such TVs are built in a manner similar to regular Android applications. Likewise, Apple TV apps are built using technologies and frameworks that are also available for iOS apps, and both types of apps can be downloaded from Apple's App Store.

Some smart TV platforms are distinct as compared to traditional Android or iOS. For example, Samsung smart TV apps are built for their own custom platform called Tizen and are downloadable from the Tizen app store. Likewise, applications for the Roku platform are built using a customized language called BrightScript, and are accessible via the Roku Channel Store. Yet another line of smart TVs such as LG smart TV and Hybrid broadcast broadband TV (HbbTV) follow a web-based ecosystem where applications are developed using HTML, CSS, and JavaScript. Finally, some smart TV platforms, such as Chromecast, do not have app stores of their own, but are only meant to "cast" content from other devices such as smartphones.

As with mobile apps, smart TV apps can integrate third-party libraries and services, often for advertising and tracking purposes. Serving advertisements is one of the main ways for smart TV platforms and app developers to generate revenue [4]. Roku's advertising revenue exceeded $250 million in 2018 and is expected to more than double by 2020 [13]. Both Roku and Fire TV take a 30% cut of the advertising revenue from apps on their platforms [14]. The smart TV advertising ecosystem mirrors many aspects of the web advertising ecosystem. Most importantly, smart TV advertising uses programmatic mechanisms that allow apps to sell their ad inventory in an automated fashion using behavioral targeting [15, 16].

The rapidly growing smart TV advertising and associated tracking ecosystem has already warranted privacy and security investigations into different smart TV platforms. Consumer Reports examined privacy policies of various smart TV platforms including Roku, LG, Sony, and Vizio [17]. They found that privacy policies are often challenging to understand and it is difficult for users to opt out of different types of tracking. For instance, many smart TVs use Automatic Content Recognition (ACR) to track their users' viewing data and use it to serve targeted ads [18]. Vizio paid $2.2 million to settle the charges by the Federal Trade Commission (FTC) that they were using ACR to track users' viewing data without their knowledge or consent [19]. While smart TV platforms now allow users to opt out of such tracking, it is not straightforward for users to turn it off [20]. Further, even with ACR turned off, users still must agree to a basic privacy policy that asks for the right to collect data about users' location, choice of apps, *etc.*

**Related Work.** While the desktop [21–23] and mobile [24–26] ATS ecosystems have been thoroughly studied, the smart TV ATS ecosystem has not been examined at scale until recently.

Three concurrent papers studied the network behavior and privacy implications of smart TVs [11, 27, 28]. Ren et al. [27] studied a large set of IoT devices, spanning multiple device categories. Their results showed that smart TVs were the category of devices that contacted the largest number of third parties, which further motivates our in-depth study of the smart TV ATS ecosystem. Huang et al. [28] used crowdsourcing to collect network traffic for IoT devices in the wild and showed that smart TVs contact many trackers by matching the contacted domains against the Disconnect blocklist. Finally, Moghaddam et al. [11] also instrumented the Roku and Fire TV platforms to map the ATS endpoints and the exposure of PII. Our work independently confirms the findings of these works w.r.t. the smart TV ATS ecosystem, both by analyzing seven

different smart TV platforms in the wild and by performing systematic tests of two platforms (Roku and Fire TV) in the lab. In addition, we further contribute along two fronts. First, we show that even the same app across different smart TV platforms contact different ATSes, which shows the fragmentation of the smart TV ATS ecosystem. Second, we evaluate the effectiveness of different sets of blocklists, including smart TV specific blocklists, in terms of their ability to prevent ads and their adverse effects on app functionality. We also suggest ways to aid blocklist curation through analysis of domain usage across apps and PII exposures.

Earlier work in this space includes [29] by Ghiglieri and Tews, who studied how broadcasting stations could track viewing behavior of users in the HbbTV platform. In contrast to the rich app-based platforms we study, the HbbTV platform studied in [29] contained only one HbbTV app that uses HTML5-based overlays to provide interactive content. Related to our work, they found that the HbbTV app loaded third-party tracking scripts from Google Analytics. Malkin et al. [30] surveyed 591 U.S. Internet users about their expectations on data collection by smart TVs. They found that users would rather enjoy new technology than worry about privacy, and users thus over rely on existing laws and regulations to protect their data.

## 2.1 Labeling Methodology

Throughout this paper, we provide insight into the smart TV ATS ecosystems by labeling a domain according to (1) its purpose (ATS or non-ATS); (2) its parent organization (*i.e.*, the domain owner); and (3) its relation to the app that uses it (first, third, or platform-specific party). We detail this methodology below.

**ATS Domains.** We identify ATS domains as follows. For figures that denote top domains, we check if the FQDN is labeled as ads or tracking by VirusTotal, McAfee, OpenDNS [31–33], or if it is blocked by any of the blocklists considered in Sec. 5. For figures and tables that involve entire datasets, we only consider the blocklists due to the impracticality of manually labeling thousands of data points.

**Parent Organizations.** To understand the presence of different organizations on smart TV platforms, we map each FQDN to its effective second level domain (eSLD) using Mozilla's Public Suffix List [34, 35], and use Crunchbase [36]'s acquisition and sub-organization information to find the parent company of the eSLD. For example, hulu.com belongs to the Walt Disney Company and youtube.com belongs to Alphabet.

**App-Level Party Categorization.** The app-level visibility in our testbed experiments (Sec. 4) enables categorization of an Internet destination as a first party or a third party w.r.t. app generating the traffic. We provide an overview of the technique here and defer details to Appendix C.1.

We adopt a technique similar to prior work [24], and we augment it to also include a platform-specific party for traffic to platform-related destinations. We match tokenized eSLDs with tokenized package/app names and developer names. If the tokens match, we label the domain as *first party*. Otherwise, if the traffic originated from platform activity rather than app activity, we label it as *platform-specific party*: for Fire TV, AntMonitor [37] labels connections with the responsible process; for Roku, we check if the eSLD contains "roku". Otherwise, if the domain is contacted by at least two different apps from different developers, we label it as *third party*. Lastly, we resort to labeling it as *other* to capture domains that are only contacted by a single app.

# 3 Smart TV Traffic in the Wild

In this section, we study the network behavior of smart TV devices when used by real users by analyzing a dataset collected at residential gateways of tens of homes (we refer to this dataset as the *in the wild* dataset). We compare the number of flows and traffic volumes generated by smart TVs from seven different platforms. We analyze the most frequently used domains of each platform by identifying ATS domains and mapping each domain to its parent organization.

**Data Collection.** To study smart TV traffic characteristics in the wild, we monitor network traffic of 41 homes in a major metropolitan area in the United States. We sniff network traffic of smart TV devices at the residential gateways using off-the-shelf OpenWRT-capable commodity routers. We collect flow-level summary information for network traffic. For each flow, we collect its start time, FQDN of the external endpoint (using DNS), and the internal device identifier. We identify smart TVs using heuristics that rely on DNS, DHCP, and SSDP traffic and also manually verify the identified smart TVs by contacting users. Our data collection covers a total of 57 smart TVs across 41 homes over the duration of approximately 3 weeks in 2018. Note that we obtained written consent from users, informing

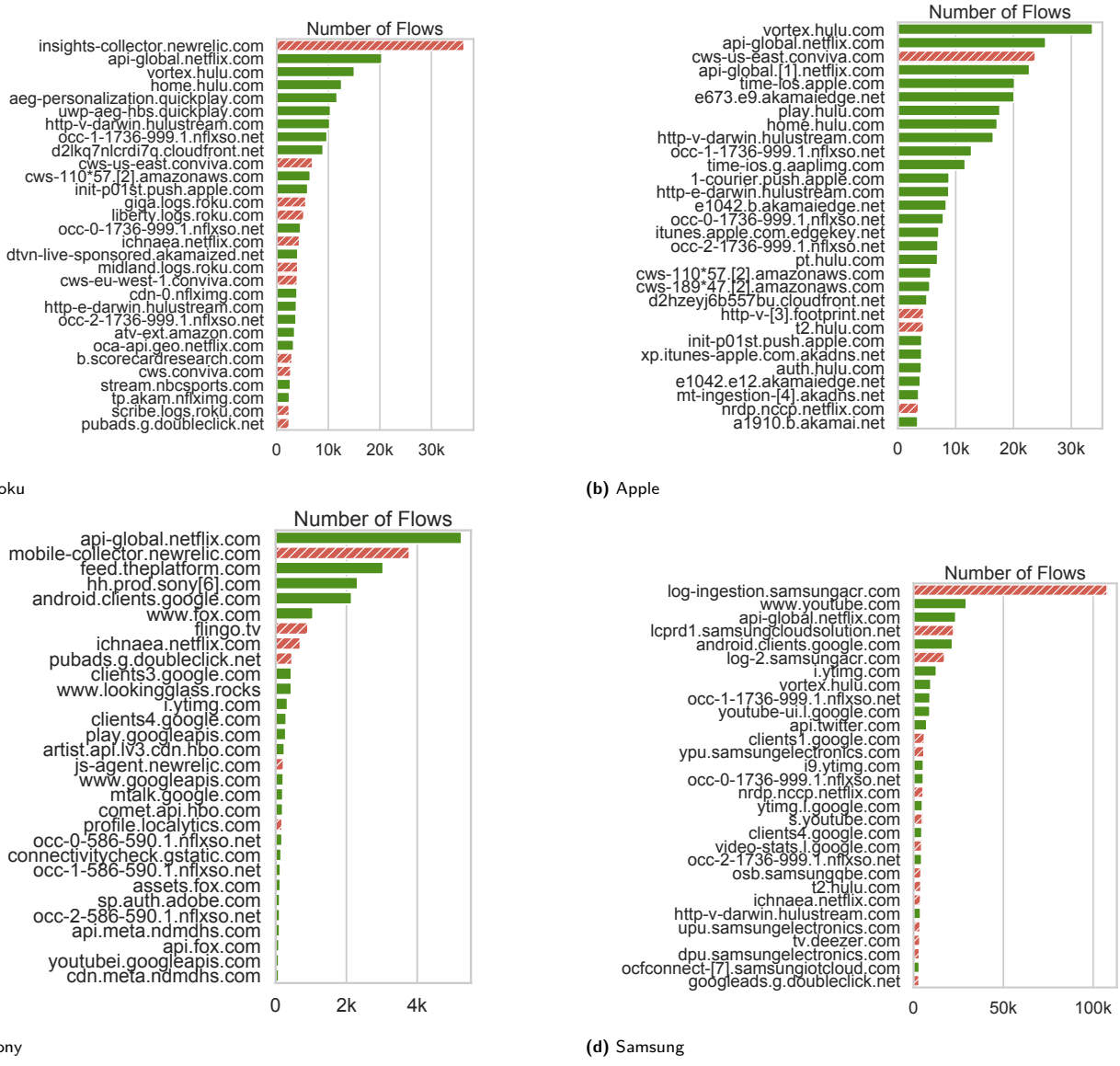



**Fig. 1.** Top-30 fully qualified domain names in terms of number of flows per device for a subset of the smart TVs in the "in the wild" dataset. See Appendix C.2 for the other brands. Domains identified as ATS are highlighted with red, dashed bars.

them of our data collection and research objectives, in accordance with our institution's IRB guidelines.

**Dataset Statistics.** Table 1 lists basic statistics of smart TV devices observed in our dataset. Overall, we note 57 smart TVs from 7 different vendors/platforms using a variety of technologies.

Devices can be built-in smart TVs such as Samsung and Sony, others like Chromecast and Apple TV can be external stick/box solutions, while devices like Roku can have both forms. For example, 7 out 9 Roku devices in our dataset were built-in Roku smart TVs, while the remaining two were external Roku sticks. Note that a smart TV platform such as Roku supports the same set of apps and a similar interface for both built-in and external smart TV devices. Thus, we do not differentiate between built-in vs. external Roku smart TV devices.

We expect smart TV devices to generate significant traffic because they are typically used for OTT video streaming [38]. Chromecast devices generate the highest number of flows (exceeding 200 thousand flows) on average, while Samsung, Apple, and Roku devices generate nearly 50 thousand flows on average. Roku devices generate the highest volume of flows (exceeding 80 GB) on average, with one Roku generating as much as 283 GB. Except for LG and Sony devices, all smart TV devices generate at least tens of GBs worth of traffic on average. Finally, we note that smart TV devices typically connect to hundreds of different endpoints on average.

| Smart TV Platform | Device Count | Average Flow Count /Device (x 1000) | Average Flow Volume /Device (GB) | Average eSLD Count /Device |
|---|---|---|---|---|
| Apple | 16 | 49.3 | 46.6 | 536 |
| Samsung | 11 | 62.6 | 33.2 | 369 |
| Chromecast | 10 | 201.9 | 26.3 | 354 |
| Roku | 9 | 48.1 | 83.0 | 543 |
| Vizio | 6 | 43.4 | 63.4 | 278 |
| LG | 4 | 10.9 | 0.9 | 1893 |
| Sony | 1 | 33.1 | 0.1 | 186 |

**Table 1.** Traffic statistics of 57 smart TV devices observed across 41 homes ("in the wild" dataset).

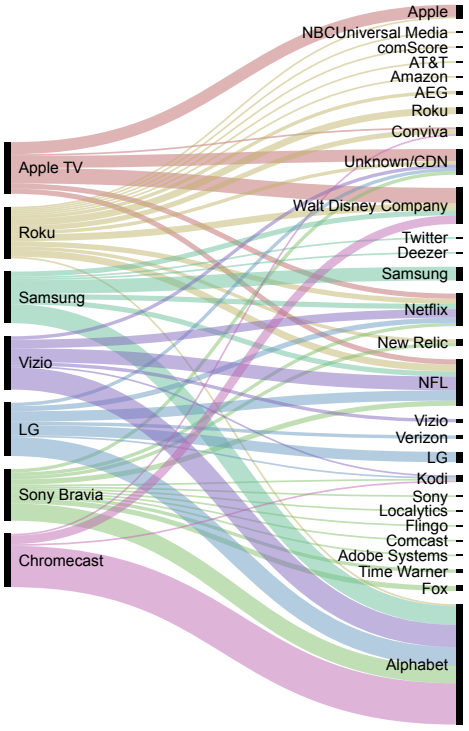



**Fig. 2.** Mapping of platforms measured in the wild to the parent organizations of the endpoints they contact (for the top-30 FQDNs of each platform). The width of an edge indicates the number of distinct FQDNs within that organization that was accessed by the platform.

**Endpoint Analysis.** Fig. 1 plots the top-30 FQDNs in terms of flow count for Roku, Apple, Sony, and Samsung smart TV platforms. The plots for the remaining smart TV platforms are in Appendix C.2.

We note several similarities in the domains accessed by different smart TV devices. First, video streaming services such as Netflix and Hulu are popular across the board, as evident from domains such api-global.netflix.com and vortex.hulu.com. Second, cloud/CDN services such as Akamai and AWS (Amazon) also appear for different smart TV platforms. Smart TVs likely connect to cloud/CDN services because popular video streaming services typically rely on third party CDNs [39, 40]. Third, we note the prevalence of well-known advertising and tracking services (ATSes). For example, *.scorecardresearch.com and *.newrelic.com are third party tracking services, and pubads.g.doubleclick.net is a third party advertising service.

We notice several platform-specific differences in the domains accessed by different smart TV platforms. For example, giga.logs.roku.com (Roku), time-ios.apple.com (Apple), hh.prod.sonyentertainmentnetwork.com (Sony), and log-ingestion.samsungacr.com (Samsung) are unique to different types of smart TVs. In addition, we notice platform-specific ATSes. For example, the following advertising-related domains are not in the top-30 (and therefore not pictured in Fig. 1), but are unique to different smart TV platforms: p.ads.roku.com (Roku), ads.samsungads.com (Samsung), and us.info.lgsmartad.com (LG).

**Organizational Analysis.** Figure 2 illustrates the mix of different parent organizations contacted by the seven smart TV platforms in our dataset. It shows the prevalence of Alphabet in smart TV platforms like Chromecast, Sony, and LG, while revealing competing organizations such as Apple on the other end of the spectrum. Furthermore, it reveals the presence of organizations like Conviva, comScore, and Localytics, whose main business is in the advertising and tracking space. We note that Samsung, Deezer, Roku, LG, and Flingo have the majority of their domains labeled as ATSes while Netflix and Walt Disney Company have less than half of their domains labeled as ATSes.

**Takeaway & Limitations.** Traffic analysis of different smart TV platforms in the wild highlights interesting similarities and differences. As expected, all devices generate traffic related to popular video streaming services. In addition, they also access ATSes, both well-known and platform-specific. While our vantage point at the residential gateway provides a real-world view of the behavior of smart TV devices, it lacks granular information beyond flows (*e.g.*, packet-level information) and does not tie traffic to the app that generate it. Another limitation of this analysis is that the findings may be biased by the viewing habits of users in these 41 households. It is unclear how to normalize to provide a fair comparison of endpoints accessed by the different smart TV platforms. We address these limitations

next by systematically analyzing two popular smart TV platforms in a controlled testbed.

# 4 Systematic Testing of the Roku and Fire TV Platforms

In this section, we perform an in-depth, systematic study of two smart TV platforms, Roku and Amazon Fire TV, which we chose since they are popular [41], affordable ($25), and among the leading smart TV platforms in terms of number of ad requests [6]. Sections 4.1 and 4.2 present our measurement approach for systematically testing the top-1000 apps in each platform while collecting their network traffic. Since app exploration is automated, no real users are involved, thus no IRB is needed. Our measurement approach provides visibility into the behaviors of individual apps, which was not possible from the vantage point used for the in the wild dataset. In Section 4.3, we analyze the two testbed datasets, and compare them to each other and to the Android ATS ecosystem.

## 4.1 Roku Data Collection

In this section, we describe the Roku platform and our app selection methodology, and present an overview of Rokustic—our software tool that automatically explores Roku apps. We use Rokustic to explore and collect traffic from 1044 Roku apps. The resulting network traces are analyzed in Sec. 4.3.

**Roku Platform.** We start by describing the Roku platform, which has its own app store—the Roku Channel Store [42] (RCS)—that offers more than 8500 apps, called "channels". For security purposes, Roku sandboxes each app (apps are not allowed to interact or access the data of other apps) and provides limited access to system resources [43]. Furthermore, Roku apps cannot run in the background. Specifically, app scripts are only executed when the user selects a particular app, and when the user exits the app, the script is halted, and the system resumes control [44].

To display ads, apps typically rely on the Roku Advertising Framework which is integrated into the Roku SDK [45]. The framework allows developers to use ad servers of their preference and updates automatically without requiring the developer to rebuild the app. Even though such a framework eliminates the need for third party ATS libraries, the development and usage of such libraries is still possible. For example, the Ooyala IQ SDK [46] provides various analytics services that can be integrated into a Roku app. Thus, such libraries can help ATSes learn the viewing habits of users by collecting data from multiple apps. In terms of permissions, Roku only protects microphone access with a permission and does not require any permission to access the advertising ID. Users can choose to reset this ID and opt out of targeted advertising at any time [45]. However, apps and libraries can easily create other IDs or use fingerprinting techniques to continue tracking users even after opt-out. We further elaborate on this in Sec 6.

**App Selection.** The RCS provides a web (and on-device) interface for browsing the available Roku apps. To the best of our knowledge, Roku does not provide public documentation on how to programmatically query the RCS. We therefore reverse-engineer the REST API backing the RCS web interface by inspecting the HTTP(S) requests sent while browsing the RCS, and use this insight to write a script that crawls the RCS for the metadata of all (8515 as of April 2019) apps.

To test the most relevant apps, we select the top-50 apps in 30 out of the 32 categories. We exclude "Themes" and "Screensavers" since these apps do not show up among the regular apps on the Roku and therefore cannot be operated using our automation software. We base our selection on the "star rating count", which we interpret as the review count. Roku apps can be labeled with multiple categories, thus some apps contribute to the top-50 of multiple categories. This places the final count of apps in our dataset at 1044.

**Automation (Rokustic).** To scale testing of apps, we implement a software tool, Rokustic, that automatically installs and exercises Roku apps. Due to lack of space, we only provide a brief overview here and defer additional details to Appendix A.1.

We run Rokustic on a Raspberry Pi that acts as a router and hosts a local wireless network that the Roku is connected to. Rokustic utilizes ECP [47], a REST-like API exposed by the Roku device, to control the Roku. Given a set of apps to exercise, Rokustic installs each app by invoking the ECP endpoint that opens up the on-device version of the RCS page for the app, and then sends a virtual key press to click the "Add Channel" button. To exercise apps, Rokustic first invokes the ECP endpoint that returns the set of installed apps. For each app, Rokustic then (1) starts tcpdump on the Raspberry Pi's wireless interface; (2) uses ECP to launch the app and invoke a series of virtual key presses in an attempt to incur content playback; (3) pauses for five minutes to

let the content play; (4) exits the app and repeats from step 2 an additional two times; (5) terminates tcpdump.

Since Roku apps cannot execute in the background (see "Roku Platform"), all captured traffic will belong to the exercised app and the Roku system. The total interaction time with each app is approximately 16 minutes. We do not attempt to decrypt TLS traffic as we cannot install our own self-signed certificates on the Roku.

| Number of | Roku | Fire TV | Both |
|---|---|---|---|
| Apps exercised | 1044 | 1010 | 128 |
| Fully qualified domain names (FQDN) | 2191 | 1734 | 578 |
| FQDNs accessed by multiple apps | 669 | 603 | 199 |
| URL paths | 13899 | 240713 | 74 |

**Table 2.** Summary of the Roku and Fire TV testbed datasets. The rightmost column summarizes the intersection between the two testbed datasets. For example, there are 128 apps that are present both in the Roku dataset and the Fire TV dataset.

## 4.2 Fire TV Data Collection

In this section, we describe the Fire TV platform, our app selection methodology, and present an overview of Firetastic—our software tool that automatically explores Fire TV apps. By using Firetastic to control six Fire TV devices in parallel, we explore and collect traffic from 1010 Fire TV apps within a one week period. The resulting network traces are analyzed in Sec. 4.3.

**Fire TV Platform.** Although Fire TV is made by Amazon, its underlying operating system, Fire OS, is a modified version of Android. This allows apps for Fire TV to be developed in a similar fashion to Android apps. Therefore, all third-party libraries that are available for Android apps can also be integrated into Fire TV apps. Similarly, application sandboxing and permissions in Fire TV are analogous to those of Android, and any permission requested by the app is inherited by all libraries that the app includes. This allows third party libraries to track users across apps using a variety of identifiers, such as Advertising ID, Serial Number, and Device ID, etc. We further discuss tracking through PII exposure in Section 6.

**App Selection.** To test the most relevant apps, we pick the top-1000 apps from Amazon's curated list of "Top Featured" apps. We ignore some apps that use a local VPN (as they would conflict with AntMonitor), that could not be installed manually, and utility apps that can change the device settings (which would affect the test environment). As a result, we ignore around 200 apps while including 1010 testable applications. Amazon's app store offers around 4,000 free apps at the time of writing, thus our dataset covers approximately 25%.

**Automation (Firetastic).** We design and implement a software tool, Firetastic, that integrates the capabilities of two open source tools for Android: an SDK for network traffic collection and a tool for input automation. We provide a brief overview of Firetastic here, and defer additional details to Appendix A.2.

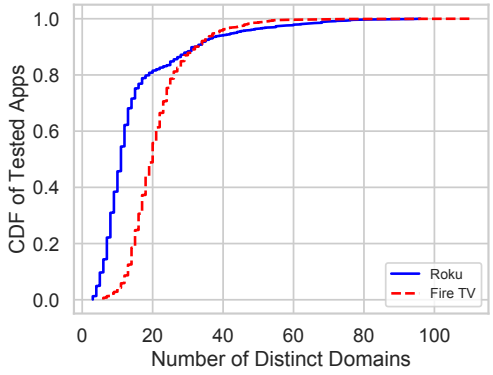
We rely on AntMonitor [37, 48], an open-source VPN-based library, to intercept all outgoing network traffic from the Fire TV, and to label each packet with the package name of the application (or system process) that generated it. We enable AntMonitor's TLS decryption for added visibility into PII exposures. We analyze the success of TLS decryption in Appendix B. In summary, TLS decryption was generally successful with 10% or fewer failures for 55% of all apps, and 20% or fewer failures for 80% of all apps.

For app exploration, we utilize DroidBot [49], a Python tool that dynamically maps the UI and simulates user inputs such as button presses using the Android Debug Bridge (ADB). To increase the probability of content playback, we configure DroidBot to utilize its breadth first search algorithm to explore each app. The intuition is that the main content is often made available from top-level UI elements.

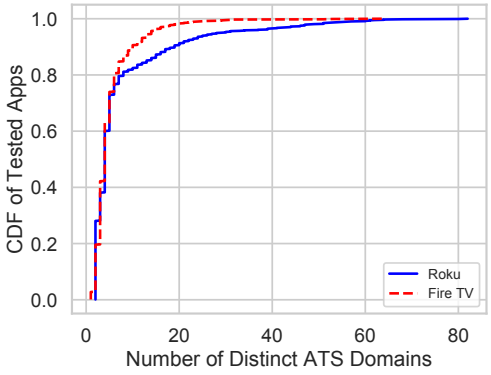
In summary, for each app, Firetastic: (1) starts AntMonitor; (2) explores the app for 15 minutes; (3) stops AntMonitor; and (4) extracts the `.pcapng` files that were generated during testing. We use Firetastic to explore apps on six Fire TV devices in parallel. Our test setup is resource-efficient and scalable, using only one computer to send commands to multiple Fire TVs.

## 4.3 Comparing Roku and Fire TV

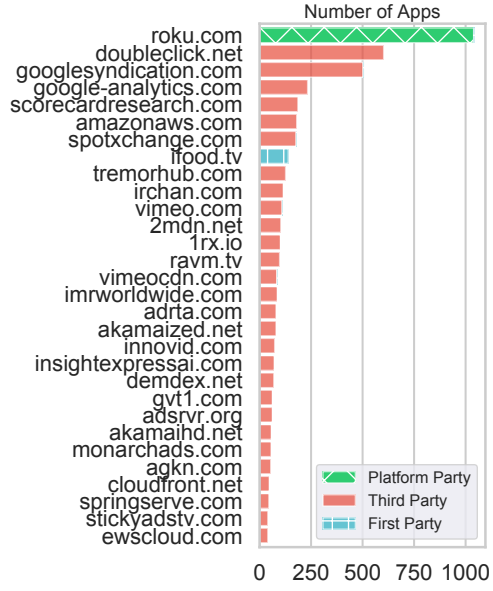
In this section, we analyze the (ATS) domains accessed by the apps in the Roku and Fire TV testbed datasets. We first provide an overview of the datasets, and analyze how many (ATS) domains apps contact. We then look closer at the eSLDs and third party ATS domains that are contacted by the most apps, including which parent organizations they belong to. Furthermore, we compare the top third party ATS domains to those of Android. Finally, we compare the domains accessed by apps that are present on both testbed platforms.



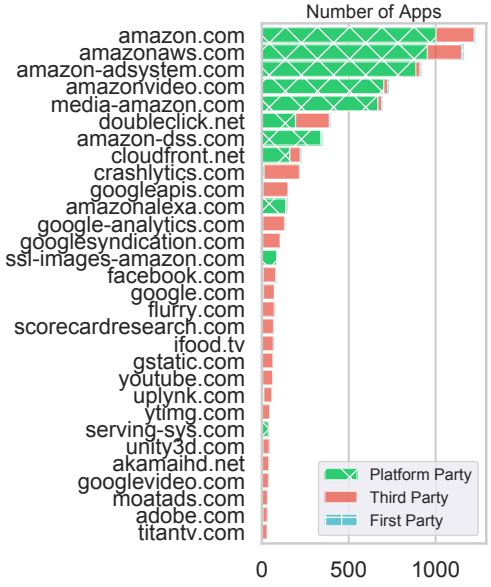

**(a) Roku & Fire TV**: Distinct domains per app.



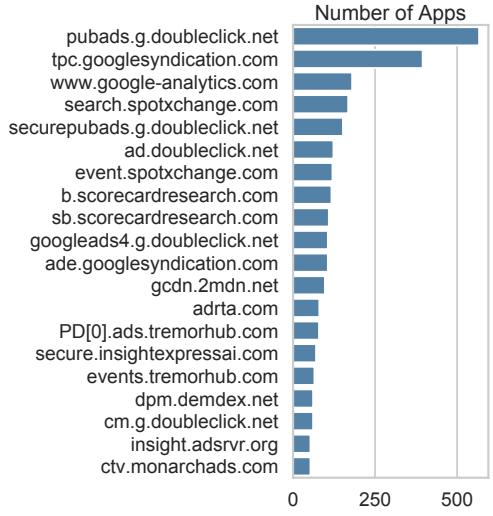

**(b) Roku & Fire TV**: Distinct ATS domains per app. A domain is considered an ATS if it is labeled as so by any of the blocklists considered in Sec. 5.



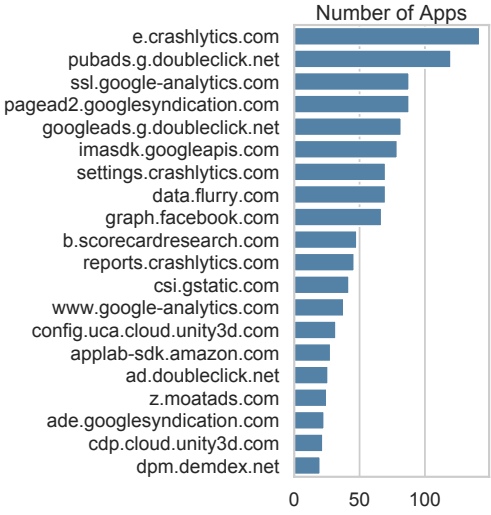

**(c) Roku**: Top-30 eSLDs.




**(d) Fire TV**: Top-30 eSLDs.




**(e) Roku**: Top-20 third party ATS domains.




**(f) Fire TV**: Top-20 third party ATS domains.

**Fig. 3.** Analysis of domain usage per app and the top domains across all apps in the Roku and Fire TV testbed datasets.

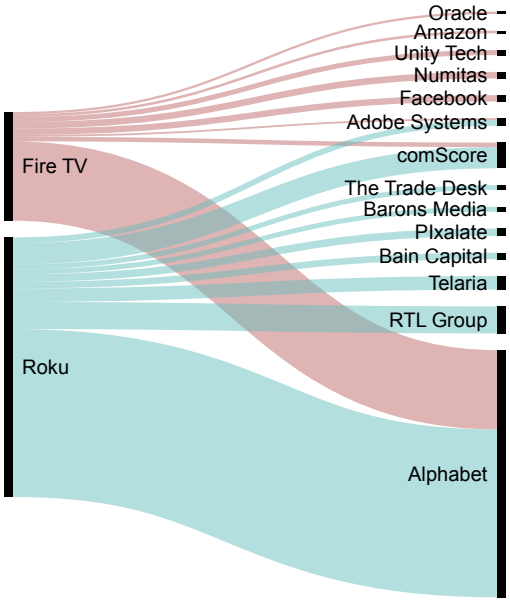



**Fig. 4.** Mapping of platforms measured in our testbed environment to the parent organizations of the top-20 *third-party* ATS domains their apps contact. The width of an edge indicates the number of apps that contact each organization.

**Overview.** The datasets collected using Rokustic and Firetastic are summarized in Table 2. For Roku, we discover 2191 distinct FQDNs, 699 of which are contacted by multiple apps. For Fire TV, we discover 1734 distinct FQDNs, 603 of which are contacted by multiple apps. We also find 578 FQDNs that appear in both datasets, 199 of which are contacted by multiple apps. Our automation uncovers approximately twice as many FQDNs as [11], possibly due to longer experiments and different app exploration goals. We further detail this in Appendix A.3.

Leveraging the blocklists from Sec. 5, we identify 314 ATS domains that are unique to the Roku dataset, 285 that are unique to the Fire TV dataset, and an overlap of 227 between the two datasets. When considering eSLDs of the ATS domains, we find 68 eSLDs that are unique to the Roku dataset, 100 that are unique to the Fire TV dataset, and an overlap of 138 eSLDs. These numbers suggest that the ATS ecosystems of the two platforms have substantial differences, which we analyze in further detail later in this section.

**Number of (ATS) Domains Contacted.** Figure 3a presents the empirical CDF of the number of distinct domains each app in the two testbed datasets contacts. Fire TV apps appear more "chatty"—most Fire TV apps contact about twice as many domains as the Roku apps. However, when we consider the number of ATS domains contacted per app in Fig. 3b, the vast majority (80%) of apps from the two platforms behave similarly.

On the positive side, for both platforms, around 60% of the apps contact only a small handful ATS domains. Yet, about 10% of the Roku and Fire TV apps contact 20+ and 10+ ATS domains, respectively. These concerning apps come from a small set of developers. For instance, for Roku, "Future Today Inc." [50], "8ctave ITV", and "StuffWeLike" [51] are responsible for 51%, 13%, and 11%, respectively, of these apps. On the Fire TV side, "HTVMA Solutions, Inc." [52] is responsible for 15% of the apps, and "Gray Television, Inc." [53] is responsible for 12% of the apps.

**Key Players.** Figures 3c (Roku) and 3d (Fire TV) present the top-30 eSLDs in terms of the number of apps that contact a subdomain of the eSLD. We define an eSLD's *app penetration* as the percentage of apps in the dataset that contact the eSLD.

*Platform.* The top eSLD of each platform has 100% app penetration and belongs to the platform operator. While these eSLDs also cover subdomains that provide functionality, we note that both platform operators are engaged in advertising and tracking, as shown in Fig. 9 in Appendix C.3, which separates traffic to the eSLDs in Figs. 3c and 3d by ATS and non-ATS FQDNs.

*Third Parties.* As evident from Figs. 3c and 3d, Alphabet has a strong presence in the ATS space of both platforms, with *.doubleclick.net, an ad delivery endpoint, achieving 58% and 35% app penetration for Roku and Fire TV, respectively. Its analytic services also rank high, with google-analytics.com in the top-20 for both platforms and crashlytics.com in the top-10 for Fire TV. To better understand additional key third party ATSes, we strip away the platform-specific endpoints and report the top-20 third party ATS FQDNs for Roku and Fire TV in Figs. 3e and 3f, respectively. We note that both platforms use distinct third party ATSes. For example, SpotX (*.spotxchange.com), which serves video ads, is a significant player in the Roku ATS space with 17% app penetration, but only maintains 1% app penetration for Fire TV. Even when considering the smaller players, we see little overlap between the two platforms, suggesting these players focus their efforts on a single platform. For example, Kantar Group's insightexpressai.com analytics service has 7% app penetration on the Roku platform, but only 0.01% on the Fire TV platform.

**Parent Organization Analysis.** We further analyze the parent organizations of Roku and Fire TV third party ATS endpoints in Fig. 4, using the method described earlier in Section 2.1. Interestingly, the set of top

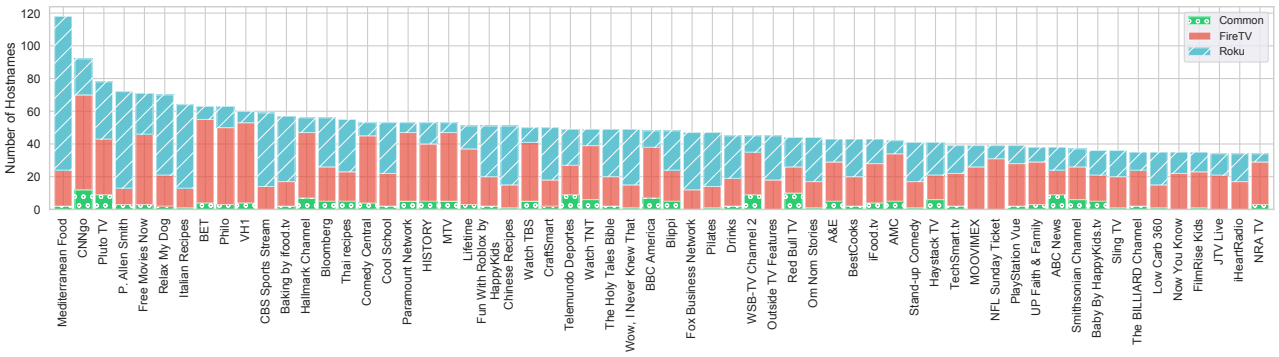



**Fig. 5.** Top-60 common apps (apps present in both testbed datasets) ordered by the number of domains that each app contacts. Considering all 128 common apps, there are 597 domains which are exclusive to Roku apps, 496 domains which are exclusive to Fire TV apps, and 155 domains which are contacted by both the Roku and the Fire TV versions of the same app.

third party organizations is rather diverse, with only a slight overlap in the shape of Adobe Systems and comScore, possibly suggesting that the remaining organizations focus their efforts on a single platform. Fire TV shows gaming and social media ATSes from Unity Tech and Facebook, whereas Roku exhibits more traffic to ATSes from companies that focus on video ads such as The Trade Desk, Telaria, and RTL Group. Similar to the in the wild organization analysis in Fig 2, we again note that Alphabet dominates the set of third party ATSes on both Roku and Fire TV.

**Comparing to Android ATS Ecosystem.** Next, we compare the top-20 third party ATS endpoints in our Roku and Fire TV datasets (Figs. 3e and 3f) with those reported for Android [24].

*Roku vs. Android.* The key third party ATSes in Roku (Fig. 3e) differ from the Android platform. For example, SpotX (*.spotxchange.com) and comScore (*.scorecardresearch.com) both have a strong presence on Roku, but are not among the key players for Android. In contrast, Facebook's graph.facebook.com is the second most popular ATS domain on Android, but insignificant on Roku. The set of top third party ATSes in Roku is also more diverse and includes smaller organizations such as Pixalate (adrta.com) and Telaria (*.tremorhub.com). While Alphabet has a strong foothold in both ATS ecosystems, it is less significant for Roku (9 out 20 ATS FQDNs are Alphabet-owned, vs. 16 out of 20 for Android).

*Fire TV vs. Android.* In contrast to Roku, Fire TV is more similar to Android: we see an overlap of 9 FQDNs, 7 of which are owned by Alphabet. This is expected, given that Fire TV is based off of Android and thus natively supports the ATS services of Android. Facebook (graph.facebook.com) and Verizon (data.flurry.com) both have a strong presence on both Fire TV and Android. Some of the third party ATS observed for Fire TV, which were not present for Android, include comScore, Adobe (dpm.demdex.net), and Amazon (applab-sdk.amazon.com).

**Common Apps in Roku and Fire TV** Next, we compare the Roku and Fire TV datasets at the app-level by analyzing the FQDNs accessed by the set of apps that appear on both platforms, referred to as *common apps*. Recall from Table 2 that the datasets collected using Rokustic and Firetastic contain a total of 128 common apps. We identified common apps by fuzzy matching app names since they sometimes vary slightly for each platform (*e.g.*, "TechSmart.tv" on Roku vs. "TechSmart" on Fire TV). We further cross-referenced with the developer's name to validate that the apps were indeed the same (*e.g.*, both TechSmart apps are created by "Future Today"). The 128 common apps contact a total of 1248 distinct FQDNs. Out of these, 597 FQDNs are exclusively contacted by Roku apps, 496 are exclusively contacted by Fire TV apps, and only 155 FQDNs are contacted by both Roku and Fire TV apps.

Figure 5 reports the amount of overlapping and non-overlapping FQDNs for the top-60 common apps (in terms of the number of distinct FQDNs that each app contacts). In general, the set of FQDNs contacted by both the Roku and the Fire TV versions of the same app is much smaller than the set of platform-specific FQDNs. From inspecting the common FQDNs for some of the apps in Fig. 5, we find that these generally include endpoints that serve content. For example, for Mediterranean Food, the only two common FQDNs are subdomains of ifood.tv, which belong to the parent organization behind the app. This makes intuitive sense as the same app presumably offers the same content on both

platforms and must therefore access the same servers to download said content. On the other hand, the platform-specific domains contain obvious ATS endpoints such as ads.yahoo.com and ads.stickyadstv.com for the Roku version of the app, and aax-us-east.amazon-adsystem.com and mobileanalytics.us-east-1.amazonaws.com for the Fire TV version of the app. In conclusion, our analysis of common apps reveals (to our surprise) little overlap in the ATS endpoints they access, which further suggests that the smart TV ATS ecosystem is segmented across platforms.

**Takeaway.** The ATS ecosystems of the Roku and Fire TV platforms seem to differ substantially: (1) the full set of ATS domains contacted by apps in the two datasets have little overlap; (2) some organizations are key players on one platform, but almost absent on the other (*e.g.*, SpotX has a significant presence on Roku, but is almost absent on Fire TV, whereas Facebook has a reasonable foothold Fire TV, but almost zero presence on Roku); and (3) apps present in both datasets have little overlap in terms of the ATS domains they contact. Finally, we find that the key third party ATS players on Android have little overlap with Roku, but substantial overlap with Fire TV, which intuitively makes sense as Fire TV is built on top of Android.

# 5 Blocklists for Smart TVs

In this section, we evaluate four well-known DNS-based blocklists' ability to prevent smart TVs from accessing ATSes and their adverse effects on app functionality. We further demonstrate how the datasets resulting from automated app exploration may aid in curating new candidate rules for blocklists.

## 5.1 Evaluating Popular DNS Blocklists

DNS-based blocking solutions such as Pi-hole [7] are used to prevent in-home devices, including smart TVs, from accessing ATS domains [54]. To block advertising and tracking traffic, they essentially "blackhole" DNS requests to known ATS domains. Specifically, they match the domain name in the DNS request against a set of blocklists, which are essentially curated hosts files that contain rules for well-known ATS domains. If the domain name is found in one of the blocklists, it is typically mapped to 0.0.0.0 or 127.0.0.1 to prevent outbound traffic to that domain [55].

| Platform | # Domains | Block Rate (%) PD | TF | MoaAB | SATV |
|---|---|---|---|---|---|
| **Dataset obtained "in the wild"** | | | | | |
| Apple | 3179 | 10% | 13% | 12% | 5% |
| Samsumg | 1765 | 14% | 19% | 15% | 8% |
| Chromecast | 1576 | 9% | 15% | 15% | 5% |
| Roku | 2312 | 15% | 19% | 18% | 7% |
| Vizio | 942 | 16% | 18% | 16% | 11% |
| LG | 627 | 45% | 54% | 50% | 27% |
| Sony | 119 | 16% | 24% | 16% | 7% |
| **Dataset obtained in our testbed** | | | | | |
| Roku | 2191 | 17% | 22% | 20% | 9% |
| Fire TV | 1734 | 22% | 27% | 22% | 9% |

**Table 3.** Block rates of the four blocklists when applied to the domains in our datasets.

**Setup.** We evaluate the following blocklists:

1. **Pi-hole Default** (PD): We test blocklists included in Pi-hole's default configuration [56] to imitate the experience of a typical Pi-hole user. This set has seven hosts files including Disconnect.me ads, Disconnect.me tracking, hpHosts, CAMELEON, MalwareDomains, StevenBlack, and Zeustracker. PD contains a total of about 133K entries.
2. **The Firebog** (TF): We test nine advertising and five tracking blocklists recommended by "The Big Blocklist Collection" [8], to emulate the experience of an advanced Pi-hole user. This includes: Disconnect.me ads, hpHosts, a dedicated blocklist targeting smart TVs, and hosts versions of EasyList and EasyPrivacy. TF contains 162K entries total.
3. **Mother of all Ad-Blocking** (MoaAB): We test this curated hosts file [9] that targets a wide-range of unwanted services including advertising, tracking, (cookies, page counters, web bugs), and malware (phishing, spyware) to again imitate the experience of an advanced Pi-hole user. MoaAB contains a total of about 255K entries.
4. **StopAd** (SATV): We test a commercial smart TV focused blocklist by StopAd [10]. This list particularly targets Android based smart TV platforms such as Fire TV. We extract StopAd's list by analyzing its APK using Android Studio's APK Analyzer [57]. SATV contains a total of about 3K entries.

We applied these blocklists to both our in the wild and testbed datasets and we report the results next.

**Block Rates.** We start our analysis by comparing how many FQDNs are blocked by the different blocklists. We define a blocklist's *block rate* as the number of distinct FQDNs that are blocked by the list, over the total

| | | App Name | No List | | PD | | TF | | MoaAB | | SATV | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | No Ads | No Breakage | No Ads | No Breakage | No Ads | No Breakage | No Ads | No Breakage | No Ads | No Breakage |
| Roku | Common | Pluto TV | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ |
| | | iFood.tv | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ |
| | | Tubi | ✓ | ✓ | ✓ | ✓ | — | ✖ | ✓ | ✓ | ✓ | ✓ |
| | Top | YouTube | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ |
| | | CBS News Live | ✗ | ✓ | ✓ | ✖ | ✓ | ✖ | ✓ | ✖ | ✓ | ✓ |
| | | The Roku Channel | ✗ | ✓ | ✓ | ✓ | ✓ | ✖ | ✓ | ✓ | ✗ | ✓ |
| | | Sony Crackle | ✗ | ✓ | ✗ | ✓ | ✓ | ✓ | ✗ | ✓ | ✗ | ✓ |
| | Random | WatchFreeComedyFlix | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✗ | ✓ |
| | | Live Past 100 Well | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | SmartWoman | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Fire TV | Common | Pluto TV | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ | ✖ |
| | | iFood.tv | ✗ | ✓ | ✓ | ✓ | — | ✖ | — | ✖ | ✓ | ✓ |
| | | Tubi | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | Top | Downloader | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | The CW for Fire TV | ✗ | ✓ | — | ✖ | — | ✖ | — | ✖ | ✗ | ✓ |
| | | FoxNow | ✗ | ✓ | — | ✖ | — | ✖ | ✗ | ✓ | ✗ | ✓ |
| | | Watch TNT | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | Random | KCRA3 Sacramento | ✗ | ✓ | ✓ | ✓ | — | ✖ | — | ✖ | ✗ | ✗ |
| | | Watch the Weather Channel | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ |
| | | Jackpot Pokers by PokerStars | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ |

**Table 4.** Missed ads and functionality breakage for different blocklists when employed during manual interaction with 10 Roku apps and 10 Fire TV apps. For "No Ads", a checkmark (✓) indicates that no ads were shown during the experiment, a cross (✗) indicates that some ad(s) appeared during the experiment, and a dash (—) indicates that breakage prevented interaction with the app altogether. For "No Breakage", a checkmark (✓) indicates that the app functioned correctly, a cross (✗) indicates minor breakage, and a bold cross (✖) indicates major breakage.

number of distinct FQDNs in the dataset. Table 3 compares the block rates of the aforementioned blocklists on our in the wild and testbed datasets. Overall, we note that TF, closely followed by MoaAB and PD, blocks the highest fraction of domains across all of the platforms in both the in the wild and testbed datasets. SATV is the distant last in terms of block rate. It is noteworthy that TF blocks more domains than MoaAB despite being about one-third shorter. We surmise this is because TF includes a smart TV focused hosts file, and thus catches more relevant smart TV ATSes. This finding shows that the size of a blocklist does not necessarily translate to its coverage.

**Blocklist Mistakes.** Motivated by the differences in the block rates of the four blocklists, we next compare them in terms of false negatives (FN) and false positives (FP). False negatives occur when a blocklist does not block requests to ATSes and may result in (visually observable) ads or (visually unobservable) PII exfiltration. False positives occur when a blocklist blocks requests that enable app functionality and may result in (visually observable) app breakage.

We first systematically quantify visually observable false negatives and false positives of blocklists by interacting with a sample of apps from our testbed datasets while coding for ads and app breakage. We sample 10 Roku apps and 10 Fire TV apps, including the top-4 free apps, three apps that are present on both platforms, and an additional three randomly selected apps. We test each app five times: once without any blocklist and four times where we individually deploy each of the aforementioned blocklists. During each experiment, we attempt to trigger ads by playing multiple videos and/or live TV channels and fast-forwarding through video content. We take note of any functionality breakage (due to false positives) and visually observable missed ads (due to false negatives). We differentiate between minor and major functionality breakage as follows: *minor breakage* when the app's main content remains available but the application suffers from minor user interface glitches or occasional freezes; and *major breakage* when the app's content becomes completely unavailable or the app fails to launch.

**Missed Ads vs. Functionality Breakage.** Table 4 summarizes the results of our manual analysis for missed ads and functionality breakage. Overall, we find that none of the blocklists are able to block ads from all of the sampled apps while avoiding breakage. In particular,

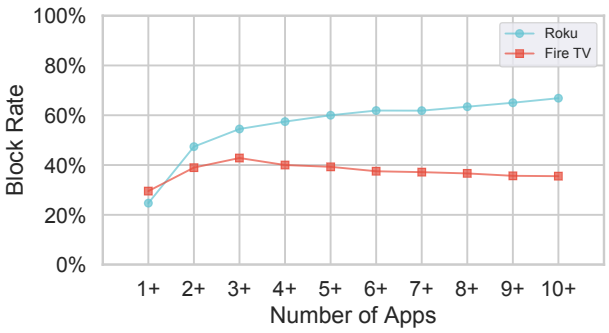



**Fig. 6.** Block rates as a function of the number apps that contact a FQDN. For the horizontal axis, "2+" represents the set of FQDNs that are contacted by 2 or more apps. For Roku, the more apps that contact an FQDN, the more likely it is that the FQDN is an ATS, according to the blocklists. The same is not true for Fire TV because platform services start to dominate the set of FQDNs that are accessed by many apps, and platform services are often not blocked.

none of the blocklists are able to block ads in YouTube and Pluto TV (available on both Roku and Fire TV). Across different lists, PD seems to achieve the best balance between blocking ads and preserving functionality.

For Roku, PD and TF perform similarly. While TF is the only list that blocks ads in Sony Crackle, both lists miss ads in YouTube and Pluto TV. TF majorly breaks three apps, while PD only majorly breaks one app. MoaAB is unable to block ads in four apps and majorly breaks only one app. SATV does not cause any breakage, but is unable to block ads in six apps.

For Fire TV, PD again seems to be the most effective at blocking ads while avoiding breakage, but is still unable to block ads in one app (Pluto TV) and majorly breaks two apps. TF is also unable to block ads in Pluto TV, but majorly breaks four apps. MoaAB is unable to block ads in three apps and majorly breaks three apps (one minor). SATV is unable to block ads in four apps and majorly breaks one app (two minor).

**Takeaway.** All blocklists suffer from a non-trivial amount of visually observable FPs and FNs. Some blocklists (*e.g.*, PD and TF) are clearly more effective than others. Interestingly, SATV, which is curated specifically for smart TVs, did not perform well.

## 5.2 Identifying False Negatives

In this section, we demonstrate how datasets generated using automation tools such as Rokustic and Firetastic enable blocklist curators to identify potential false negatives in the blocklist. In particular, we observe that the more apps that contact a FQDN, the more likely it is that the FQDN is an ATS. This is intuitive and consistent with a similar observation previously made in the mobile ecosystem [24].

We first use simple keywords such as "ad", "ads", and "track" to shortlist obvious ATS domains in our datasets. While keyword search is not perfect, this simple approach identifies several obvious false negatives (we provide the full list in Appendix C.4). For example, p.ads.roku.com and adtag.primetime.adobe.com are advertising/tracking related domains which are not blocked by any of the lists.

We observe that many of these false negatives (i.e., missed ATS domains) are contacted by multiple apps in our testbed datasets. For example, p.ads.roku.com is accessed by more than 100 apps in our Roku testbed dataset. To gain further insight into potential false negatives, we study whether the likelihood of being blocked is impacted by the number of apps that access a domain. Figure 6 plots the block rates for the union of the four blocklists as a function of FQDNs' occurrences across apps in our testbed datasets. We note that the block rate substantially increases for the domains that appear across multiple apps. For example, the block rate almost doubles for domains contacted by two or more apps as compared to domains contacted by a single or more apps. Domains that are contacted by multiple different apps are therefore more likely to belong to third party ATS libraries included by smart TV apps.

# 6 PII Exposures in Smart TVs

In this section, we examine our testbed datasets from Sec. 4 for exposure of personally identifiable information (PII) and we evaluate the effectiveness of blocklists in preventing it. We define "PII exposure" as the transmission of any PII from the smart TV device to any Internet destination. We identify PII values (such as advertising ID and serial number) through the settings menus and packaging of each device. Since trackers are known to encode or hash PII [58], we compute the MD5 and SHA1 hashes for each of the PII values. We then search for these PII values in the HTTP header fields and URI path. Recall from Section 4 that we can analyze HTTP information even for encrypted flows in the Fire TV dataset due to AntMonitor's TLS decryption [37, 48], but can only analyze unencrypted flows in Roku. The number of PII exposures reported for Roku should therefore be considered a lower bound.

| PII | Roku Testbed Dataset (Apps & eSLDs) | | | | Fire TV Testbed Dataset (Apps & eSLDs) | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1st Party | 3rd Party | Other | Total | 1st Party | 3rd Party | Platform Party | Other | Total |
| Advertising ID | 4/4/25% | 263/36/88% | 6/3/0% | 269/42/81% | 17/7/25% | 53/31/78% | 715/4/71% | 5/5/40% | 725/39/71% |
| Serial Number | 48/17/5% | 128/16/74% | 2/2/0% | 174/34/36% | 10/3/0% | 51/4/33% | 867/4/9% | 2/2/0% | 881/9/12% |
| Device ID | - | - | - | - | 19/8/0% | 153/27/36% | 819/5/14% | 10/11/21% | 856/43/31% |
| Username | 4/4/0% | 1/1/100% | - | 5/5/20% | 1/2/0% | 2/2/100% | 1/1/100% | - | 4/5/40% |
| MAC | - | - | - | - | - | 2/2/100% | - | - | 2/2/100% |
| Location | - | 42/2/100% | - | 42/2/100% | - | 27/7/90% | 2/2/100% | - | 28/7/90% |

**Table 5. Applications / eSLDs / % Distinct FQDNs Blocked.** Number of apps that expose PII, number of distinct eSLDs that receive PII from these apps, and percentage of distinct subdomains of the eSLDs that are blocked by the blocklists. We further separate by party as defined in Sec. 2.1. Roku platform column omitted since we do not observe PII exposures to platform domains.

**Overview.** Table 5 reports the PII exposures for both testbed platforms. For Roku, the majority of the PII exposures are to third parties, whereas for Fire TV they are to the platform-specific party. The blocklists adequately prevent exfiltration of PII to third parties, blocking 74% or more of third party domains for all the PII values considered for Roku. For Fire TV, they mitigate the majority of advertising ID exposures, blocking 71% or more of the involved domains, but are not as effective in preventing exposures of serial number and device ID to third parties and platform destinations.

**Differentiating PII Exposures.** Inspired by [59], we adopt a simple approach for distinguishing between "good" and "bad" PII exposures that treats PII exposures to third parties as a higher threat to privacy than PII exposures to first parties. PII exposures to first parties are generally warranted as they likely have a functional purpose such as personalization of content (*e.g.*, keeping track of where the user paused a video). For example, the Roku app "Acacia Fitness & Yoga Channel" from "RLJ Entertainment", sends a request to the first party domain api.rlje.net with a URI path of "/cms/acacia/today/roku/content/browse.json" while including the device's serial number in an HTTP header field, suggesting that the serial number is used to personalize today's featured content.

On the other hand, PII exposures to third parties are generally unwarranted as they typically do not have a functional purpose. This extends to cases where the app retrieves its content through a third party CDN as the personalization could be achieved by first sending the PII to the first party server which could then respond to the app with the CDN URL for the content to be retrieved. For instance, the Roku app "ArmchairTourist" from "ArmchairTourist Video Inc." sends a request to the third party domain ads.adrise.tv with a URI path of "/track/impression..." that encodes the device's serial number, suggesting that the PII is used to track what ads have been shown to the user.

**Exposures to Platform Party.** For Fire TV, the majority of the exposures of serial number and device ID to platform destinations seem to be for advertising and tracking purposes. For example, 697 apps send the serial number and device ID (and advertising ID) to the platform endpoint aviary.amazon.com with a URI path of "/GetAds", and 53 apps send the serial number to dna.amazon.com, with a URI path of "/GetSponsoredTileAds". Judging from these paths, it would seem like the advertising ID alone would be sufficient and the more appropriate PII. On the other hand, some exposures seem to serve a functional purpose. For example, 67 apps send the serial number to atv-ext.amazon.com, with varying URI paths containing "/cdp/". We surmise that this domain serves as "Content Delivery Platform(s)" [60], allowing apps to personalize content without user login. Specifically, we see paths such as "/cdp/playback/GetDefaultSettings" coupled with an "x-atv-session-id" HTTP header field.

**Joint Exposure of Static and Dynamic PII.** We observe that some apps send the advertising ID *alongside* other static identifiers. This goes against recommended developer practices, where apps and ATSes should only rely on dynamic identifiers that can be refreshed like advertising ID to give users the ability to opt out of being tracked. Aside from the 697 Fire TV apps that expose advertising ID alongside serial number and device ID discussed earlier, we observe 10 Roku apps (including prominent ones such as Pluto TV and PBS) that send both the advertising ID and the serial number to third parties (subdomains of scorecardreasearch.com and youboranqs01.com). Similarly, 12 Fire TV apps send the advertising ID alongside the device ID to third party destinations such as ads.adrise.tv and ctv.monarchads.com. Thus, these

practices allow ATSes to link an old advertising ID to the new value by joining on the static identifiers.

**Leveraging Missed PII Exposures to Improve Blocklists.** The above indicate another direction for improving blocklist curation for smart TVs. By deploying tools such as Rokustic and Firetastic and searching the network traces for PII exposures, blocklist curators can generate candidate rules that can then be examined manually. Using this approach, we identified 38 domains in the Roku dataset and 30 in the Fire TV dataset that receive PII, but were not blocked by any list. These numbers are conservative as we exclude location and account name that are likely to be used for legitimate purposes, such as logging in or serving location-based content. These domains include obvious ATSes such as ads.aimitv.com and ads.ewscloud.com. Another noteworthy mention is hotlist.samba.tv: Samba TV uses Automatic Content Recognition to provide content suggestions on smart TVs, but this comes at the cost of targeted advertising that even propagates onto other devices in the home network [61].

**Takeaway.** Hundreds of Roku and Fire TV apps expose PII, mostly to third parties and the platform-specific party. For Fire TV, we observe that most of the exposures to the platform-specific party seem to be for advertising and tracking purposes. We observe that many Roku and Fire TV apps send the advertising ID *alongside* a static identifier (*e.g.*, serial number), which enables the ATS to relink a user profile associated with an old advertising ID to a new advertising ID, thus eliminating the user's ability to opt out. The blocklists generally do well at preventing exposure of the advertising ID on both platforms, but are less successful at preventing exposures of serial number and device ID on Fire TV.

# 7 Conclusion & Directions

**Summary.** In this paper, we performed one of the first comprehensive measurement studies of the emerging smart TV advertising and tracking service (ATS) ecosystem. To that end, we analyzed and compared: (i) a realistic but small *in the wild* dataset (57 smart TV devices from seven different platforms, with coarse flow-level information); and (ii) two large *testbed* datasets (top-1000 apps on Roku and Fire TV, tested systematically, with granular per app and packet-level information). Our work establishes that the smart TV ATS ecosystem is fragmented across different smart TV platforms and is different from the mobile ATS ecosystem. We further evaluate popular DNS-based blocklists' ability to prevent smart TVs from accessing ATSes, and find that all lists suffer from missed ATSes and incur app breakage. Finally, we examine our testbed datasets for exposure of personally identifiable information (PII) and discover that hundreds of apps send PII to third parties and the platform-specific party, mostly for advertising and tracking purposes.

**Limitations.** Our methodology has its limitations. First, the automated app exploration may not always result in content (video/audio) playback, which may impact the (ATS) domains discovered. We evaluate the extent of this limitation in Appendix A.3. We find that Rokustic and Firetastic perform on par with concurrent work [11] in terms of playback success, and that they manage to discover a large fraction of the number of domains discovered during manual interaction. Second, apps may prevent TLS interception through use of certificate pinning, which may prevent Firetastic from observing PII exposures in encrypted traffic. We assess the decryption failures in Appendix B: we find that for 80% of the apps in our Fire TV testbed dataset, TLS interception only fails for 20% or fewer of an app's TLS connections. Third, our analysis of FPs and FNs in DNS-based blocklists in Sec. 5.1 does not account for DNS over HTTPS (DoH), nor static advertisements, thus it may overcount blocklist FNs for these cases.

**Future Work.** Our findings motivate more research to further understand smart TVs and to develop privacy-enhancing solutions specifically designed for each smart TV platform. For example, more research is needed to curate accurate, fine-grained (as opposed to DNS-based), and platform-specific blocklists. To foster further research along this direction, we plan to make our tools, Rokustic and Firetastic, and testbed datasets publicly available [12]. We intend to further improve our tools' ability to thoroughly explore smart TV apps along the directions discussed in Appendix A.3.

# Acknowledgements

This work is supported in part by NSF Awards 1715152, 1750175, 1815131, and 1815666, Seed Funding by UCI VCR, and Minim. The authors would like to thank M. Hammad Mazhar and the team at Minim for their help with collecting and analyzing smart TV traffic in the wild. We would also like to thank our shepherd, Erik Sy, and the anonymous PETS reviewers for their feedback that helped significantly improve the paper.

# References


[1] Rimma Kats. How Many Households Own a Smart TV? https://www.emarketer.com/content/how-many-households-own-a-smart-tv, 2018. [Online; accessed 2019-05-10].

[2] Hulu gained twice as many US subscribers as Netflix at the start of 2019. https://www.cnbc.com/2019/05/01/hulu-gained-twice-as-many-subscribers-as-netflix-in-us.html, 2019. [Online; accessed 2019-05-10].

[3] Amazon: Smart TVs. https://www.amazon.com/smart-tv-store/b?ie=UTF8&node=5969290011, 2019. [Online; accessed 2019-05-10].

[4] Connected TV Advertising is Surging. https://www.videonuze.com/article/connected-tv-advertising-is-surging, 2017. [Online; accessed 2019-05-10].

[5] MAGNA ADVERTISING FORECASTS. https://magnaglobal.com/magna-advertising-forecasts-fall-update-executive-summary/, 2019. [Online; accessed 2019-05-10].

[6] Jump PR. Beachfront Releases 2018 CTV Ad Data, Roku Still Leads, Amazon Growing Quickly. https://www.broadcastingcable.com/post-type-the-wire/2018-ctv-ad-data-realeased-by-beachfront, 2018. [Online; accessed 2019-05-10].

[7] Pi-Hole: A black hole for Internet advertisements. https://pi-hole.net/, 2019. [Online; accessed 2019-05-11].

[8] WaLLy3K. The Big Blocklist Collection. https://firebog.net, 2019. [Online; accessed 2019-04-29].

[9] MoaAB: Mother of All AD-BLOCKING. https://forum.xda-developers.com/showthread.php?t=1916098, 2019. [Online; accessed 2019-04-22].

[10] Kromtech Alliance Corp. Stopad for tv. https://stopad.io/tv, 2019.

[11] 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*, CCS '19, pages 131–147, New York, NY, USA, 2019. ACM.

[12] UCI Networking Group. The TV is Smart and Full of Trackers: Project Page. http://athinagroup.eng.uci.edu/projects/smarttv/, 2019.

[13] Ross Benes. 10 Ways Roku Is Growing Its Ad Business. https://www.emarketer.com/content/10-ways-roku-is-growing-its-ads-business, 2019. [Online; accessed 2019-05-10].

[14] Garett Sloane. AMAZON IS NOW TAKING A 30 PERCENT CUT OF AD SALES FROM FIRE TV. https://adage.com/article/design/amazon-taking-30-percent-ad-sales-fire-tv/315678, 2018. [Online; accessed 2019-05-10].

[15] Roku, Inc. The Roku Advantage. https://advertising.roku.com/advertising-solutions, 2019. [Online; accessed 2019-05-10].

[16] Amazon.com, Inc. Amazon DSP. https://advertising.amazon.com/products/amazon-dsp, 2019. [Online; accessed 2019-05-10].

[17] "Consumer Reports". Samsung and Roku Smart TVs Vulnerable to Hacking. https://www.consumerreports.org/televisions/samsung-roku-smart-tvs-vulnerable-to-hacking-consumer-reports-finds/, 2018. [Online; accessed 2019-04-22].

[18] Sapna Maheshwari. How Smart TVs in Millions of U.S. Homes Track More Than What's On Tonight. https://www.nytimes.com/2018/07/05/business/media/tv-viewer-tracking.html, 2018. [Online; accessed 2019-05-10].

[19] "FTC". VIZIO to Pay $2.2 Million to FTC, State of New Jersey to Settle Charges It Collected Viewing Histories on 11 Million Smart Televisions without Users' Consent. https://www.ftc.gov/news-events/press-releases/2017/02/vizio-pay-22-million-ftc-state-new-jersey-settle-charges-it, 2017. [Online; accessed 2019-04-22].

[20] Whitson Gordon. How to Stop Your Smart TV From Tracking What You Watch. https://www.nytimes.com/2018/07/23/smarter-living/how-to-stop-your-smart-tv-from-tracking-what-you-watch.html, 2018. [Online; accessed 2019-05-10].

[21] J. R. Mayer and J. C. Mitchell. Third-Party Web Tracking: Policy and Technology. In *2012 IEEE Symposium on Security and Privacy*, pages 413–427, May 2012.

[22] Phillipa Gill, Vijay Erramilli, Augustin Chaintreau, Balachander Krishnamurthy, Konstantina Papagiannaki, and Pablo Rodriguez. Follow the Money: Understanding Economics of Online Aggregation and Advertising. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 141–148. ACM, 2013.

[23] Steven Englehardt and Arvind Narayanan. Online Tracking: A 1-million-site Measurement and Analysis. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1388–1401, New York, NY, USA, 2016. ACM.

[24] 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. *NDSS*, 2018.

[25] Jingjing Ren, Ashwin Rao, Martina Lindorfer, Arnaud Legout, and David Choffnes. ReCon: Revealing and Controlling PII Leaks in Mobile Network Traffic. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pages 361–374. ACM, 2016.

[26] Anastasia Shuba, Athina Markopoulou, and Zubair Shafiq. NoMoAds: Effective and Efficient Cross-App Mobile Ad-Blocking. *Proceedings on Privacy Enhancing Technologies*, 2018(4):125–140, 2018.

[27] 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*, IMC '19, pages 267–279, New York, NY, USA, 2019. ACM.

[28] Danny Yuxing Huang, Noah Apthorpe, Gunes Acar, Frank Li, and Nick Feamster. IoT Inspector: Crowdsourcing Labeled Network Traffic from Smart Home Devices at Scale, 2019.

[29] M. Ghiglieri and E. Tews. A privacy protection system for HbbTV in Smart TVs. In *2014 IEEE 11th Consumer Com-*

*munications and Networking Conference (CCNC)*, pages 357–362, Jan 2014.

[30] 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 *European Workshop on Usable Security (Euro USEC)*, 2018.

[31] VirusTotal. https://www.virustotal.com/. [Online; accessed 2019-08-24].

[32] McAfee, LLC. Customer URL Ticketing System. https://www.trustedsource.org/. [Online; accessed 2019-08-24].

[33] OpenDNS Domain Tagging. https://community.opendns.com/domaintagging/. [Online; accessed 2019-08-24].

[34] Mozilla Foundation. Public Suffix List. https://publicsuffix.org/. [Online; accessed 2019-08-23].

[35] John Kurkowsi. tldextract. https://github.com/john-kurkowski/tldextract. [Online; accessed 2019-08-23].

[36] Crunchbase. https://www.crunchbase.com/. [Online; accessed 2019-08-29].

[37] Anastasia Shuba, Anh Le, Emmanouil Alimpertis, Minas Gjoka, and Athina Markopoulou. AntMonitor: A System for On-Device Mobile Network Monitoring and its Applications. *arXiv preprint arXiv:1611.04268*, 2016.

[38] Jeffrey Erman, Alexandre Gerber, KK Ramadrishnan, Subhabrata Sen, and Oliver Spatscheck. Over The Top Video: The Gorilla in Cellular Networks. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 127–136. ACM, 2011.

[39] Timm Böttger, Felix Cuadrado, Gareth Tyson, Ignacio Castro, and Steve Uhlig. Open Connect Everywhere: A Glimpse at the Internet Ecosystem through the Lens of the Netflix CDN. *ACM SIGCOMM Computer Communication Review*, 48(1):28–34, 2018.

[40] Vijay Kumar Adhikari, Yang Guo, Fang Hao, Volker Hilt, and Zhi-Li Zhang. A tale of three CDNs: An active measurement study of Hulu and its CDNs. In *2012 Proceedings IEEE INFOCOM Workshops*, pages 7–12. IEEE, 2012.

[41] eMarketer. US Connected TV Users, by Brand, 2018 & 2022. https://www.emarketer.com/Chart/US-Connected-TV-Users-by-Brand-2018-2022-of-connected-TV-users/220767, 2018. [Online; accessed 2019-11-26].

[42] Roku, Inc. Roku Channel Store. https://channelstore.roku.com, 2019. [Online; accessed 2019-04-19].

[43] Roku, Inc. Roku Developer Documentation: Security Overview. https://sdkdocs.roku.com/display/sdkdoc/Security+Overview, 2019. [Online; accessed 2019-04-22].

[44] Roku, Inc. Roku Developer Documentation: Development Environment Overview. https://sdkdocs.roku.com/display/sdkdoc/Development+Environment+Overview, 2019. [Online; accessed 2019-04-22].

[45] Roku, Inc. Roku Developer Documentation: Roku Advertising Framework. https://sdkdocs.roku.com/display/sdkdoc/Roku+Advertising+Framework, 2019. [Online; accessed 2019-04-22].

[46] Ooyala IQ SDK for Roku. https://github.com/ooyala/iq-sdk-roku, 2015. [Online; accessed 2019-04-22].

[47] Jeff Bush, Kevin Cooper, and Linda Kyrnitszke. External Control API. https://sdkdocs.roku.com/x/K5cY, 2013. [Online; accessed 2019-03-04].

[48] AntMonitor open-source. https://github.com/UCI-Networking-Group/AntMonitor, 2018. [Online; accessed 2019-05-10].

[49] Yuanchun Li, Ziyue Yang, Yao Guo, and Xiangqun Chen. DroidBot: a Lightweight UI-guided Test Input Generator for Android. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 23–26. IEEE, 2017.

[50] Future Today Inc. http://www.stuffwelike.com/, 2019. [Online; accessed 2019-12-05].

[51] StuffWeLike. http://www.stuffwelike.com/, 2019. [Online; accessed 2019-12-05].

[52] Manta Media Inc. Htvma Solutions, Inc. https://www.manta.com/c/mhqfv38/htvma-solutions-inc, 2019. [Online; accessed 2019-12-05].

[53] Gray Television, Inc. https://gray.tv/, 2019. [Online; accessed 2019-12-06].

[54] telekrmor. Round 3: What Really Happens On Your Network? https://pi-hole.net/2017/07/06/round-3-what-really-happens-on-your-network/, 2017. [Online; accessed 2019-05-11].

[55] Pi-hole LLC. Blocking Mode. https://docs.pi-hole.net/ftldns/blockingmode, 2018.

[56] Customising Sources for Ad Lists. https://github.com/pi-hole/pi-hole/wiki/Customising-Sources-for-Ad-Lists, 2019.

[57] Google LLC. apkanalyzer. https://developer.android.com/studio/command-line/apkanalyzer, 2019. [Online; accessed 2019-04-29].

[58] Steven Englehardt, Jeffrey Han, and Arvind Narayanan. I never signed up for this! privacy implications of email tracking. *Proceedings on Privacy Enhancing Technologies*, 2018(1):109–126, 2018.

[59] 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.

[60] Antidot. Content Delivery Platform. https://www.antidot.net/content-delivery-platform/, 2019. [Online; accessed 2019-12-05].

[61] Sapna Maheshwari. How Smart TVs in Millions of U.S. Homes Track More Than What's On Tonight. https://www.nytimes.com/2018/07/05/business/media/tv-viewer-tracking.html, 2018. [Online; accessed 2019-05-11].

[62] Raspberry Pi Foundation. Setting up a Raspberry Pi as an access point in a standalone network (NAT). https://www.raspberrypi.org/documentation/configuration/wireless/access-point.md, 2019. [Online; accessed 2019-03-04].

[63] Android tcpdump. https://www.androidtcpdump.com/, 2019. [Online; accessed 2019-04-11].

# Appendix

In appendices A and B, we discuss implementation details and limitations of our methodology, and outline directions for improvement. In Appendix C, we provide additional analysis of datasets that had to be omitted from the main text due to lack of space. Notably, the rest of the "in the wild" dataset is within Appendix C.2.

# A Automatic App Exploration

In this section, we provide details on the implementation of our automatic app exploration tools, Rokustic for Roku (Sec. A.1) and Firetastic for Fire TV (Sec. A.2), in addition to what is described in the main paper (Sections 4.1 and 4.2, respectively). Then, in Section A.3, we discuss the limitations of Rokustic and Firetastic, including when the automation does not lead to video playback and how the automated exploration compares to manual testing of apps that require login. Ultimately, this discussion leads to directions for how to improve Rokustic's and Firetastic's automatic app exploration.

## A.1 Roku Automation

To scale testing of apps, we implement a software system, Rokustic, that automatically installs and exercises Roku apps on a Roku Express 3900X (Roku for short).

**Setup.** We run Rokustic on a Raspberry Pi 3 Model B+ set up to host a standalone network as per the instructions given in [62]. The Pi's wireless interface (`wlan0`) is configured as a wireless access point with DHCP server and NAT, and the Roku is connected to this local wireless network. The Pi's wired interface (`eth0`) connects the Pi and thus, in turn, the Roku to the WAN. This setup enables us to collect all traffic going in and out of the Roku by running `tcpdump` on the Raspberry Pi's wireless interface. We do not attempt to decrypt TLS traffic as we cannot install our own self-signed certificates on the Roku.

Rokustic utilizes the Roku External Control (ECP) API [47] to control the Roku. The ECP is a REST-like API exposed by the Roku to other devices on the local network. The ECP includes a set of REST-endpoints that provides the ability to press keys on the Roku remote, query the Roku for various information such as the set of installed apps, programmatically browse the Roku Channel Store (RCS), *etc.*

**App Installation.** Given a set of apps to exercise, Rokustic installs each app by invoking the ECP endpoint that opens up the on-device version of the RCS page for the app, and then sends a virtual key press to click the "Add Channel" button that starts download and installation of the app. Rokustic then waits for five seconds, and then queries the Roku for the set of installed apps to check if the installation has completed, and if so continues to install the next app, otherwise the wait-and-check is repeated (until a fixed threshold).

**App Exploration.** From manual inspection of a few apps (*e.g.* YouTube and Pluto TV), we find that playable content is often presented in a grid, where each cell is a different video or live TV channel. Generally, the user interface defaults to highlighting one of these cells (*e.g.*, the first recommended video). Pressing "Select" on the Roku remote immediately after the app has launched will therefore result in playback of some content. From this insight, we devised a simple algorithm (see Listing 1) that attempts to cause playback of three different videos for each installed Roku app.

```
for app in roku.installedApps:
        startPacketCapture(app.id);
        # Play default video
        launch(app); sleepSeconds(20);
        press("SELECT"); sleepMinutes(5);
        # Play some other video by selecting a
        # different cell in the grid
        relaunch(app);
        press("DOWN"); press("DOWN");
        press("RIGHT"); press("RIGHT");
        press("SELECT"); sleepMinutes(5);
        # Play a 3rd video
        relaunch(app);
        press("DOWN"); press("DOWN");
        press("SELECT"); sleepMinutes(5);
        # Quit the Roku app
        press("HOME");
        stopPacketCapture(app.id);
```

**Listing 1.** Algorithm for exercising Roku apps.

For each Roku app, the algorithm first starts a packet capture so as to produce a `.pcap` file for each Roku app, thereby essentially labeling traffic with the app that caused it: since Roku apps cannot execute in the background (see Sec. 4.1), all traffic captured during execution of a single app will belong to that app and the Roku system. The target app is launched, and the algorithm pauses, waiting for the app to load. Next, a virtual "Select" key press is sent to attempt to start

video playback, and the algorithm subsequently pauses to let the content play. The app is then relaunched by returning to the Roku's home screen and then launching the app again. The purpose of this is to safely return to the app's home screen such that different content can be selected for playback. This is repeated two times, with slight variations in the sequence of navigational key presses, such that each app will (presumably) end up playing three different videos, making the total exploration time approximately 16 minutes per app. We note that the `relaunch` procedure internally performs short sleeps to let the app quit and launch again, and that we have omitted a one second sleep after each navigational key press in the pseudo code in Listing 1.

Although the Roku remote has a "Back" button, which behaves similarly to the back button in Android, we purposefully avoid using it as a means to return to the app's home screen: if the video selected for playback is shorter than the sleep duration, the app will return to its home screen automatically, and pressing "Back" will therefore quit the app and return to the Roku's home screen. This would pollute our data as the subsequent navigational key presses would cause a different app to be highlighted and then launched by the next "Select" key press.

## A.2 Fire TV Automation

Since Fire TV is based on Android, we can use existing Android tools to capture network traffic. Although there are various methods for capturing traffic on Android on the device itself (*e.g.* `androidtcpdump` [63]), most of them require a rooted device. While it is possible to root a Fire TV, it may make applications behave differently if they detect root. Thus, to collect measurements that are representative of an average user, we use a VPN-based traffic interception method that does not require rooting the device [37, 48]. We discard incoming traffic because video content results in huge pcap files, which due to a technical limitation of ADB (very slow transfer speeds for large files) slows down the automated experiments significantly. The outgoing traffic is sufficient for our domain and PII analysis.

To automatically explore each Fire TV application, we utilize Droidbot [49], as it treats each app as a tree of possible paths to explore instead of randomly generating events, which results in higher test coverage of the application. Furthermore, we deduce that developers would minimize the necessary clicks in order to reach the core sections of their applications, especially for playing video content. Thus, we configure DroidBot to utilize its breadth first search algorithm to explore each application. The intuition is that this should cover more distinct UI paths of the app, thus increasing the chance of content playback (in contrast, the Depth First Search algorithm may cause the automation to deep end into a path that we do not care about, such as a settings menu). With some trial and error, we selected the input command interval as three seconds which leaves enough time for applications to handle the command and load the next view during app exploration.

We summarize Firetastic's automation algorithm in Listing 2. For each app, Firetastic first starts the local VPN to capture (and decrypt) traffic. Next, it invokes DroidBot, which in turn launches the app and begins exploring it. When the 15-minute exploration completes, Firetastic stops the local VPN and extracts the `.pcapng` files that were generated during testing.

```
device = "10.0.1.xx:5555"
pcapng_dir = "/some/path/to/store/pcapng"
apk_dir = "/some/path/to/apk/batch"
for app in apk_dir:
        # Start AntMonitor on Fire TV
        start_antmonitor(device)
        # Ensure VPN connection up
        ensure_antmonitor_connected(device)
        # Run DroidBot command
        params = { duration: 15min,
                   policy: "bfs_naive",
                   install_timeout: 5min,
                   interval: 3sec }
        run_droidbot(app, params, device)
        # Stop AntMonitor on Fire TV
        stop_antmonitor_vpn(device)
        # Extract the pcap files
        extract_pcapng_files(app, pcapng_dir,
                             device)
        # Clean up before testing next app
        remove_pcapng_files(device)
```

**Listing 2.** Algorithm for exercising FireTV apps.

## A.3 Limitations of App Exploration

Our automated app exploration has, admittedly, limitations. For example, it can miss video playback for some apps, and cannot fully explore apps that require login. In this section, we evaluate our automated exploration vs. a more realistic manual exploration by a real user,

| | | App Name | Playback? (auto) | Distinct eSLDs Auto (A) | Manual (M) | A/M | Distinct ATS domains A | M | A/M |
|---|---|---|---|---|---|---|---|---|---|
| Roku | Top-10 | YouTube | ✓ | 8 | 15 | 53% | 5 | 15 | 33% |
| | | Sling TV | ✗ | 10 | 21 | 48% | 6 | 21 | 29% |
| | | The Roku Channel | ✓ | 16 | 13 | 123% | 7 | 5 | 140% |
| | | Crackle | ✓ | 9 | 34 | 26% | 8 | 42 | 19% |
| | | JW Broadcasting | ✗ | 3 | 4 | 75% | 2 | 2 | 100% |
| | | PBS KIDS | ✓ | 4 | 7 | 57% | 4 | 6 | 67% |
| | | ESPN | ✗ | 6 | 16 | 38% | 4 | 16 | 25% |
| | | Tubi - Free Movies & TV | ✗ | 3 | 19 | 15.79% | 5 | 34 | 15% |
| | | DisneyNOW | ✗ | 6 | 9 | 67% | 4 | 5 | 80% |
| | | Pluto TV - It's Free TV | ✓ | 24 | 9 | 267% | 36 | 7 | 514% |
| | Random | Roku Newscaster | ✗ | 4 | 5 | 80% | 3 | 2 | 150% |
| | | ChuChu TV | ✓ | 10 | 10 | 100% | 9 | 17 | 53% |
| | | Elvis | ✓ | 20 | 38 | 53% | 14 | 54 | 26% |
| | | Mondo | ✗ | 4 | 5 | 80% | 2 | 2 | 100% |
| | | tik tok | ✓ | 1 | 3 | 33% | 2 | 3 | 67% |
| | **Total** | | 8 of 15 (53%) | 74 | 113 | 65% | 64 | 122 | 52% |
| Fire TV | Top-10 | Pluto TV - It's Free TV | ✓ | 15 | 30 | 50% | 13 | 35 | 37% |
| | | ABC | ✓ | 14 | 13 | 108% | 5 | 6 | 83% |
| | | Fox Now | ✓ | 22 | 24 | 92% | 18 | 18 | 100% |
| | | AMC | ✗ | 18 | 28 | 64% | 8 | 19 | 42% |
| | | Fox Sports GO | ✗ | 12 | 19 | 63.16% | 7 | 17 | 41% |
| | | Kids for Youtube | ✓ | 16 | 19 | 84% | 7 | 5 | 140% |
| | | PBS Kids | ✓ | 12 | 15 | 80% | 7 | 9 | 78% |
| | | CNN Go | ✓ | 31 | 34 | 91% | 41 | 34 | 121% |
| | | Sundance TV | ✗ | 13 | 23 | 57% | 5 | 11 | 45% |
| | | MTV | ✗ | 56 | 38 | 147% | 38 | 54 | 70% |
| | Random | Vimeo | ✓ | 12 | 11 | 109% | 5 | 3 | 167% |
| | | Dog TV Online | ✓ | 10 | 14 | 71% | 2 | 5 | 40% |
| | | WCSC Live 5 News | ✗ | 13 | 12 | 108% | 5 | 5 | 100% |
| | | WFXG FOX 54 | ✓ | 18 | 18 | 100% | 8 | 7 | 114% |
| | | 13abc WTVG Toledo, OH | ✓ | 12 | 11 | 109% | 5 | 2 | 250% |
| | **Total** | | 10 of 15 (67%) | 125 | 117 | 107% | 115 | 138 | 83% |

**Table 6.** Content playback success for Rokustic and Firetastic, and a comparison of the number of domains discovered by Rokustic and Firetastic to the number of domains discovered during manual interaction with the same app, for 15 apps that do *not* require login. For each app, we perform approximately 16 minutes of automated and 16 minutes of manual interaction.

| | App Name | Distinct eSLDs Auto (A) | Manual (M) | A/M | Distinct ATS domains A | M | A/M |
|---|---|---|---|---|---|---|---|
| Roku | HBO NOW | 3 | 7 | 43% | 2 | 5 | 40% |
| | Hulu | 6 | 18 | 33% | 3 | 21 | 14% |
| | Netflix | 3 | 4 | 75% | 3 | 3 | 100% |
| | SHOWTIME | 4 | 8 | 50% | 3 | 6 | 50% |
| | STARZ | 5 | 7 | 71% | 3 | 5 | 60% |
| | **Total** | 14 | 30 | 47% | 6 | 26 | 23% |
| Fire TV | HBO NOW | 7 | 9 | 78% | 2 | 2 | 100% |
| | Hulu | 9 | 19 | 47% | 4 | 17 | 24% |
| | Netflix | 11 | 9 | 122% | 4 | 3 | 133% |
| | SHOWTIME | 10 | 8 | 125% | 4 | 3 | 133% |
| | STARZ | 18 | 14 | 129% | 12 | 7 | 171% |
| | **Total** | 27 | 42 | 64% | 15 | 27 | 56% |

**Table 7.** Comparison of the number of domains discovered by Rokustic and Firetastic to the number of domains discovered during manual interaction with the same five apps that *require* login. The automation was performed while logged out, and the manual interaction was performed while logged in. For each app, we perform approximately 16 minutes of automated and 16 minutes of manual interaction.

for 20 apps. We report the cases where our automatic exploration succeeds and fails, and we provide insights into the reasons why, and ideas for future improvements.

First, we evaluate our tools' ability to perform playback, because it affects our ability to capture traffic related to ads delivered at the start of or during content (video/audio) playback. We evaluate Firetastic's and Rokustic's content playback rates in Sec. A.3.1 by performing a case study of 15 apps. We find that the two tools perform similarly to the the state-of-the-art [11], and we identify how we can further improve the tools to increase the chance of incurring content playback.

In Sec. A.3.2, we evaluate the automated approach's ability to discover an app's domain space by comparing the eSLDs and ATS domains discovered by the tools to the eSLDs and ATS domains discovered during manual interaction with the same 15 apps. We find that both tools generally do well at mapping a large fraction of the number of domains discovered during manual interaction.

Finally, a common limitation of app exploration in general (not just for smart TV apps) is that it is difficult to explore apps that require user login. In Sec. A.3.3 we compare the eSLDs and ATS domains discovered by Firetastic and Rokustic for 5 popular streaming apps (while logged out) to the eSLDs and ATS domains discovered during manual interaction (while logged in). As expected, we find that the automation misses parts of the domain space of apps that serve third-party ads as part of content that is only accessible after logging in. Interestingly, we also observe cases where the automation discovers more (ATS) domains than the manual experiments.

### A.3.1 Video Playback

**Setup.** To evaluate Firetastic's and Rokustic's ability to incur playback of an app's main content (video/audio) for the set of apps that do not require login to access (parts of) their content, we run the full (∼16 minutes) automation for 15 apps on each platform, while observing for content playback. We pick the 15 apps according to their popularity (as defined in Sec. 4): (1) the top-10 most popular apps to capture the most influential apps; and (2) 5 additional randomly sampled apps, spread evenly across the popularity spectrum, to represent the dataset as a whole.

**Results.** We present the content playback results in Table 6. Firetastic manages to play content for 67% of the 15 apps (60% for the top-10 apps, and 80% for the random apps). A common characteristic of the 33% unsuccessful apps is that the majority of their content is locked, and that free content is deferred to the least accessible sections of the UI (*e.g.*, the bottom row of a grid layout). This decreases the chance that Firetastic will discover a playable video during the 16 min experiment, especially since attempts to play locked content often redirects to login/activation screens where additional time is lost. We can improve on this in future work by mixing BFS and DFS exploration: the automation can explore each level up to a certain threshold before drilling down into the next nested view.

Rokustic manages to play content for 53% of the 15 apps (50% for the top-10 apps, and 60% for the random apps). Unsuccessful attempts are primarily due to nested menus, where additional "Select" key press(es) are necessary to start playback. Through manual interaction with a few apps prior to executing our top-1000 apps measurement, we observe that if an app starts playback on the first "Select" key press, additional key presses may result in pausing and/or exiting the video, or skipping past an ad. Therefore, we opt for a conservative approach that, when successful, will collect as much ad/video traffic as possible. In future work, we can further improve content playback by repeatedly sending "Select" key presses (with short pauses in between) until the network throughput for the Roku stays above a certain threshold for a short duration (*e.g.*, the bitrate of 720p video for 5 seconds). This dynamic strategy can handle more nested menus and thus be resilient to future changes to an app's UI.

**Takeaway.** In summary, both tools work well when an app adheres to good UI design principles, such as reducing the number of user actions required to reach the main content. The content playback success rates are on par with state-of-the-art concurrent work [11], with Firetastic slightly ahead, possibly due to its dynamic approach to UI exploration (as opposed to the static, heuristic-based approach used in [11]), and with Rokustic slightly behind. While Firetastic leverages existing advanced Android tools (Droidbot), similar tools do not currently exist for Roku, thus we had to build them from scratch, and it is therefore natural that Rokustic falls slightly behind its Fire TV counterpart.

### A.3.2 Automation vs. Manual Testing

**Setup.** Next, we evaluate Firetastic's and Rokustic's ability to successfully map an app's domain space. We manually interact with the 15 apps from Sec. A.3, and compare the network behavior observed during automated testing to the network behavior observed during manual testing. For a fair comparison, we interact with each app for approximately the same duration as in the automated experiments (∼16 minutes). For consistency, we follow a protocol in which we attempt to play 7 different videos for approximately 2 minutes each, leaving a few minutes to navigate between videos. We compare the network behavior in terms of the number of eSLDs and ATS domains (as defined by the union of the blocklists from Sec. 5) contacted by each app. The results are presented in Table 6.

**Results.** Firetastic is successful in mapping the domain space for 10 out of 15 apps (67%), uncovering 0.8 times (or more) the number of eSLDs, and 0.7 times (or more) the number of ATS domains discovered in the manual experiments. In fact, Firetastic even discovers *more* eSLDs and ATS domains than the manual experiment for 6 (40%) and 7 (47%), respectively, of the 15 apps. Rokustic is less successful, but still manages to uncover 0.67 times (or more) the number of eSLDs and ATS domains in the manual experiment for 7 (47%) and 8 (53%), respectively, of the 15 apps. Moreover, Rokustic even discovers 2.67 times as many eSLDs as the manual experiment for one of the apps (Pluto TV).

**Intuition.** The two tools have very different approaches to app exploration: Firetastic seeks to explore as much app functionality as possible, but is likely to exit content playback early, whereas Rokustic seeks to mimic a real user that sits through 3 videos of 5 minutes. Each approach has its own merit: Firetastic is good at discovering many ATS domains for apps that present ads *before* content playback begins, whereas Rokustic is the more successful tool when it comes to discovering ATS domains for apps that defer ad delivery to later in the video/audio stream, as was the case for Pluto TV. Finally, we note that even in the worst case, *i.e.*, when Firetastic and Rokustic do not manage to incur content playback, they still uncover several ATS domains.

**Takeaway.** This case study, and the fact that Firetastic and Rokustic uncovered approximately twice as many domains as the state-of-the-art [11] for the top-1000 apps measurement described in Sec. 4, show that our tools already provide sufficient means to automatically estimate a lower bound on the ATS domains of the two platforms. This lower bound should improve when we implement the changes suggested earlier.

### A.3.3 Apps that Require Login

**Setup.** To understand how well the automation manages to map the domain spaces of apps that require login, we pick 5 of the top subscription-based streaming apps and run the automation *without* logging in, and also manually interact with the same apps *while logged in* (following the same protocol as in Sec. A.3.2). We compare the network behavior using the same metrics as in Sec. A.3.2. The results are presented in Table 7.

**Results.** Firetastic actually discovers *more* eSLDs than the manual experiments for 3 out of the 5 apps. We also observe that the STARZ app contacts *more* ATS domains when automatically tested than in the manual experiment. These findings are interesting as they indicate that an app's domain space and ATS-related activity possibly changes after the user logs in and is not necessarily tied to video playback. An ideal experiment would thus need to thoroughly exercise the app in both states. The results for Rokustic are more in line with what is to be expected: Rokustic discovers fewer eSLDs and ATS domains than the manual experiments (55% and 53%, respectively, on average). Finally, for both platforms, we observe that the number of ATS domains contacted by Hulu increases significantly for the manual experiments. This is to be expected as Hulu is the only of the 5 apps that deliver third-party ads during content playback.

**Takeaway.** As expected, Rokustic can only map parts of the (ATS) domain spaces of apps that require login. For Firetastic, we observe that some apps contact *more* ATS domains while logged out, and an ideal experiment would thus need to explore the apps in both states.

## B Fire TV TLS Interception

Firetastic attempts to decrypt TLS traffic to facilitate detection of PII exposures in encrypted traffic. However, the decryption may fail: (i) for apps that attempt to mitigate TLS interception, for example through use of certificate pinning, or (ii) if the cipher suites used in the TLS connection are not supported by the TLS decryption library used in AntMonitor. To approximate the impact that such decryption failures may have on the PII exposure results, we evaluate the failure rate

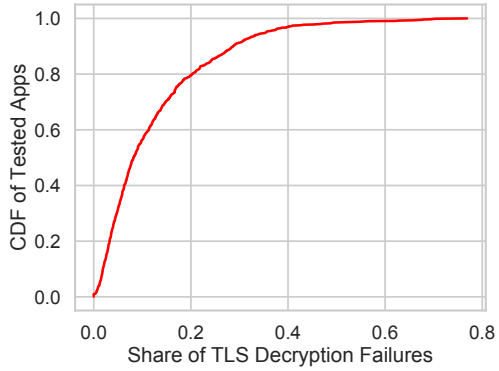



**Fig. 7.** Empirical CDF of TLS decryption failures per app. The decryption generally worked well: Decryption fails for 1 out of 10 (or fewer) TLS connections for 55% of all apps; 1 out of 5 (or fewer) TLS connections for 80% of all apps.

of Firetastic's TLS interception across all apps in the Fire TV testbed dataset from Sec. 4.

**Methodology.** For each app in the Fire TV testbed dataset, we first identify the set of TCP connections, $t$, initiated by this app and labeled as TLS by tshark. Next, we identify the subset $h$ of TCP connections in $t$ that also contained at least one packet labeled by tshark as HTTP: these are the connections that are successfully decrypted. Finally, we compute the decryption failure rate of each app as $\frac{|t|-|h|}{|t|}$. We note that our methodology conservatively computes an upper bound compared to the actual failure rate, as any non-HTTP over TLS (*e.g.*, proprietary binary protocols) will be counted as decryption failures.

We note that in $t$, we only include TLS connections, where the TLS handshake concluded successfully. We assume that an app will retry the connection if it rejects AntMonitor's certificate. AntMonitor stops intercepting an app's connections if it detects that the app rejects its certificate, thus the second TLS handshake should complete successfully. This restriction on $t$ therefore also prevents double counting (*i.e.*, the original failed connection does not contribute to the total number of TLS connections initiated by the app). Although we only recorded upstream data for Fire TV, we use the presence of an upstream TLS Application Data packet as a proxy for inferring that the TLS handshake concluded successfully.

**Results.** In Fig. 7, we show the empirical CDF for the TLS decryption failure rates for all apps in the Fire TV testbed dataset. We note that the decryption generally works well. For example, decryption fails for 1 out of 10 (or fewer) TLS connections for 55% of all apps, and 1 out of 5 (or fewer) TLS connections for 80% of all apps. Since TLS decryption was generally successful, this validates the PII exposure results.

# C Additional Analysis of Datasets

## C.1 Labeling Datasets Continued

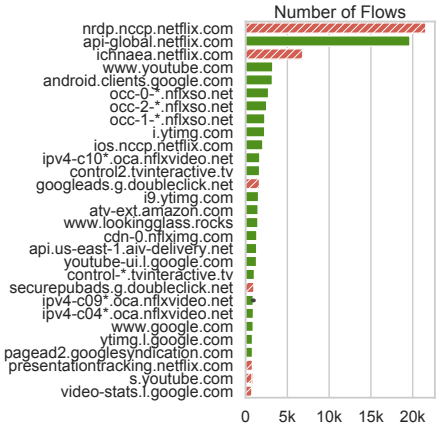
To complement Sec 2.1, this section provides the full details for how we label each endpoint as either first party, third party or platform-specific party, w.r.t. the app that contacts it:

1. We first tokenize app identifiers and the eSLD of the contacted FQDN (we obtain the eSLD using Mozilla's Public Suffix List [34, 35]). For Fire TV, we tokenize the package names and developer names. For Roku, we rely on app and developer names since its apps do not have package names. For app/package tokens, we ignore common and platform-specific strings like "com", "firetv", "roku", etc., while retaining all tokens from the developer names. We then match the resulting identifiers with the tokenized eSLD.
2. If the tokens match, we label the destination as *first party*. We note that since we keep all developer tokens, we will map "roku" related eSLDs as first party if the app was developed officially by Roku.
3. Otherwise, we label a destination as *platform-specific party* if it originated from platform activity rather than app activity. For Fire TV, we rely on AntMonitor's [37] ability to label each connection with the responsible process. For Roku, we simply check if the eSLD contains "roku".
4. Otherwise, if the destination is contacted by at least two different apps from different developers, we label it as *third party*.
5. Finally, if the destination does not fall into any of the other categories, we resort to labeling it as *other*, which thus captures domains that are only contacted by a single app and are not identified as a first party nor platform-specific party.

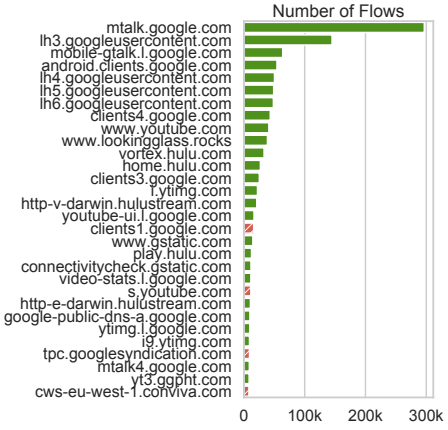
We acknowledge that the variations in labeling methodology for platform-specific parties for Roku and Fire TV may impact comparability. However, we believe that our choice provides the more accurate platform-specific labeling for each testbed platform.

## C.2 In the Wild Dataset Continued

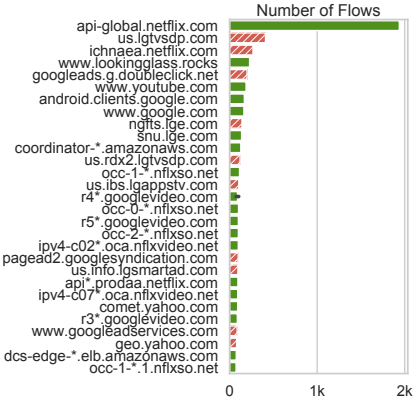
Figure 8 complements Fig. 1 in Sec. 3, providing data for the remaining three devices.




**(a)** Vizio




**(b)** Chromecast




**(c)** LG

**Fig. 8.** Continuation of Fig. 1 from Sec. 3: Top-30 fully qualified domain names in terms of number of flows per device for the remaining devices in the in the wild dataset. Domains identified as ATSes are highlighted with red, dashed bars.

## C.3 Key Players Continued

Figure 9 provides further insight into the "Keys Players" discussion in Sec. 4.3 by accumulating flows to subdomains of the top-30 eSLDs of each platform. Each eSLD is in turn mapped to its parent organization. Finally, all flows to domains under that parent organization are separated based on whether they are labeled as advertising and tracking or not. A large amount of the flows to the two platform operators are labeled as advertising and tracking.

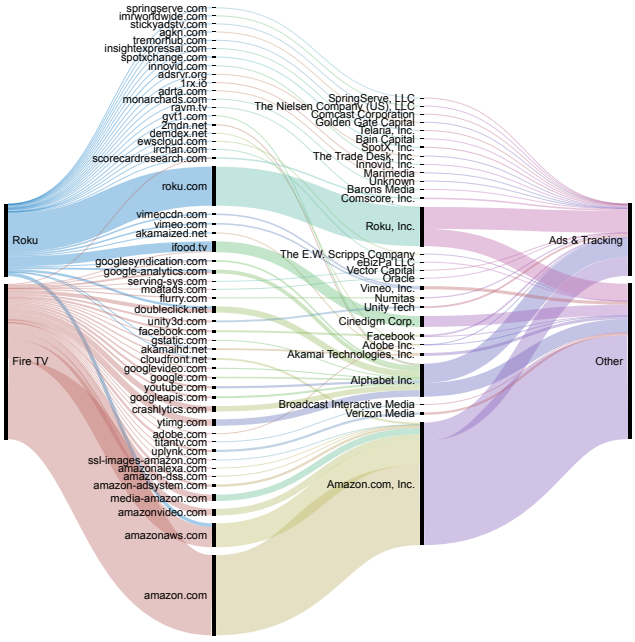



**Fig. 9.** Left to right: (1) mapping of flows to subdomains of the top-30 eSLDs (see Figs. 3c and 3d) of each testbed platform; (2) mapping from eSLD to its parent organization; (3) separation of the flows to the organization by advertising and tracking flows and other flows. An edge's width represents the number of flows.

## C.4 False Negatives for Blocklists

Table 8 provides a full list of false negatives to complement Sec 5.2. We discover potential ATS domains by looking at whether multiple apps contact the domain and through keyword searches.

| **Hostname** | PD | TF | MoaAB | SATV |
|---|---|---|---|---|
| p.ads.roku.com | ✗ | ✗ | ✗ | ✗ |
| ads.aimitv.com | ✗ | ✗ | ✗ | ✗ |
| adtag.primetime.adobe.com | ✗ | ✗ | ✗ | ✗ |
| ads.adrise.tv | ✗ | ✓ | ✗ | ✗ |
| ads.samba.tv | ✗ | ✓ | ✗ | ✗ |
| tracking.sctv1.monarchads.com | ✗ | ✓ | ✗ | ✗ |
| ads.ewscloud.com | ✗ | ✗ | ✗ | ✗ |
| trackingrkx.com | ✗ | ✗ | ✗ | ✗ |
| us-east-1-ads.superawesome.tv | ✗ | ✗ | ✗ | ✗ |
| track.sr.roku.com | ✗ | ✗ | ✗ | ✗ |
| router.adstack.tv | ✗ | ✗ | ✗ | ✗ |
| metrics.claspws.tv | ✗ | ✗ | ✗ | ✗ |
| customerevents.netflix.com | ✗ | ✓ | ✗ | ✗ |
| event.altitude-arena.com | ✗ | ✓ | ✗ | ✗ |
| ads.altitude-arena.com | ✗ | ✓ | ✗ | ✗ |
| myhouseofads.firebaseio.com | ✗ | ✗ | ✗ | ✗ |
| mads.amazon.com | ✗ | ✗ | ✗ | ✗ |
| ads.aimitv.com.s3.amazonaws.com | ✗ | ✗ | ✗ | ✗ |
| analytics.mobitv.com | ✗ | ✗ | ✗ | ✗ |
| events.brightline.tv | ✗ | ✗ | ✗ | ✗ |
| ctv.monarchads.com | ✗ | ✓ | ✗ | ✗ |
| ads.superawesome.tv | ✗ | ✓ | ✗ | ✗ |
| adplatform-static.s3-us-west-1.amazonaws.com | ✗ | ✗ | ✗ | ✗ |
| kraken-measurements.s3-external-1.amazonaws.com | ✗ | ✗ | ✗ | ✗ |
| kinstruments-measurements.s3-external-1.amazonaws.com | ✗ | ✗ | ✗ | ✗ |
| venezia-measurements.s3-external-1.amazonaws.com | ✗ | ✗ | ✗ | ✗ |
| ad-playlistserver.aws.syncbak.com | ✗ | ✗ | ✗ | ✗ |

**Table 8.** Examples of potential false negatives for the four DNS-based blocklists found using app penetration analysis and keywords search (“ad”, “ads”, “analy”, “track”, “hb” (for heartbeat), “score”, “event”, “metrics”, “measure”).