UNIVERSITY OF CALIFORNIA,
IRVINE

Characterizing the Fire TV Advertising and Tracking Ecosystem

THESIS

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Computer Engineering

by

Hieu Le

Thesis Committee:
Professor Athina Markopoulou, Chair
Assistant Professor Salma Elmalaki
Associate Professor Zubair Shafiq

2021

DEDICATION

I dedicate this to my parents, Hong Nguyen and Doan Le, who were immediately supportive of me going to graduate school. Thank you for always encouraging me to reach for the stars.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank my advisor, Athina Markopoulou, for guiding me through my first published research project. Thank you for being the main reason for me to undertake graduate school. Your patience and understanding for your students — especially for me — know no bounds.

Special thanks to Zubair Shafiq, for your advice and instruction on how to approach the smart TV research space and graduate life in general.

Thank you to my committee member, Salma Elmalaki, for continuous research interactions on future work.

Thank you to my co-authors, Janus Varmarken and Anastasia Shuba — your positivity, determination, and strength, inspirit me throughout this project.

This work is supported in part by NSF Awards 1715152, 1750175, 1815131, 1815666, 1956393, Seed Funding by UCI VCR, and UCI EECS Fellowship.

Thank you to PETS for allowing authors to retain the copyright to their work. Portion of this thesis' text is a reprint of the material as it appears in Janus Varmarken, Hieu Le, Anastasia Shuba, Athina Markopoulou, and Zubair Shafiq: "The TV is Smart and Full of Trackers: Measuring Smart TV Advertising and Tracking", Proceedings on Privacy Enhancing Technologies, 2020.

ABSTRACT OF THE THESIS

Characterizing the Fire TV Advertising and Tracking Ecosystem

By

Hieu Le

Master of Science in Computer Engineering

University of California, Irvine, 2021

Professor Athina Markopoulou, Chair

Popular smart TV platforms present advertising and tracking opportunities for the platform provider, app developers, and third party organizations. In this thesis, we conduct one of the first large-scale and systematic study of advertising and tracking services (ATS) on Amazon's Fire TV — a platform that is part of 100 million devices sold worldwide. To do so, we build Firetastic, a tool that can automatically explore and collect network traffic for Fire TV apps. Through exploration of 1K apps, we identify 512 unique ATS domains — 60% of apps contact three to six of these domains. We conclude that ATS is prevalent on Fire TV. The ecosystem is dominated by its platform provider, Amazon, with a large third party presence from Alphabet. It is comparable to Android, sharing many common third party ATS but is more diverse and mature than the Oculus VR platform. Furthermore, we consider four state-of-the-art DNS-based blocklists that users can employ to block ATS. We observe that none of the blocklists are effective at blocking ads without causing some form of functionality or visual breakage. In addition, they fail to block the majority of personal identifiable information (PII) exposures such as Device ID and Serial Number to third party.

Chapter 1

Introduction

Smart TVs are commonplace in households worldwide, providing consumers with Internet-based features through apps for video streaming (*e.g.* Netflix, Disney Plus), gaming (*e.g.* Candy Crush Saga), and utility (*e.g.* calculator, alarm clock). They come in various platforms such as Amazon's Fire TV, Roku, Apple TV, etc... With large consumer bases, these platforms provide advertising opportunities within apps, streaming video content, and even in the dashboard itself [2]. In addition, platform providers get a share of the ad revenue; *e.g.* Amazon gets a 30% cut for Fire TV [36]. However, these ads and tracking services (ATS), on popular smart TV platforms, are not well understood by users and researchers.

Amazon's Fire TV, in particular, has sold over 100 million devices with 50 million active users and is among the leaders in ad requests [1, 2, 16] . By 2021, the platform comes as either external dongles (with five variations of Fire TV dongles ranging from \$18 – \$80) or built into the TV itself (with 18 smart TV listings on Amazon with Fire TV ranging from \$170 – \$1500) [3, 4]. Due to its prevalence, we focus on Amazon's Fire TV, and provide one of the first large-scale and systematic study of its ATS ecosystem with two main contributions:

1. Tool for Automatic Data Collection. We create Firetastic [18], an open-sourced tool

that automatically explores Fire TV apps and collect network traffic. It is scalable by using one machine to instrument multiple Fire TV devices. With Firetastic, we explore 1010 apps in a testbed setting and provide this large network traffic dataset to the public [11].

2. Evaluation of Fire TV ATS Ecosystem. We investigate Fire TV's ATS ecosystem at different vantage points to answer whether ATS is prevalent on the platform. In particular, we consider: (1) which personal identifiable information (PII) are exposed to ATS (*e.g.* Advertising ID, Serial Number, Device ID); (2) in which context is information being exposed (first party, third party, and platform-specific party); and (3) the parent organization that owns these ATS. We find that PII are exposed to all parties, but especially to third parties. The ecosystem is dominated by Amazon (the platform provider), and has Alphabet as the main third party. We conclude that ATS is prevalent on Fire TV — 60% of apps contact three to six different ATS, while 10% of apps contact 10–20 different ATS. Furthermore, we evaluate whether users can protect their privacy by blocking ATS without affecting their user experience. We consider four popular DNS-based blocklists that users can utilize and observe that they only block 33% and 36% of Serial Number and Device ID exposures to third party, respectively. In addition, we rely on breakage analysis (*i.e.* whether the app is still functional or visually intact) to determine whether the blocklists disrupt the user experience. Our findings show that blocklists were unable to block the majority of ads without causing some breakage, motivating the need for smart TV specific blocklists. Thus, we leverage our large dataset to recommend ways to identify ATS based on keywords.

Overview. In Chapter 2, we provide an overview of the Fire TV platform and app store, and discuss related work. Chapter 3 details the design of Firetastic, including how it collects network traffic and explores apps automatically. Next, Chapter 4 provides our evaluation of Fire TV's ATS ecosystem and how it compares against other platforms. Chapter 5 evaluates the effectiveness of DNS-based blocklists in blocking ads without breakage and preventing PII exposures. Lastly, Chapter 6 concludes and provides future directions.

Chapter 2

Background & Related Work

2.1 Background

2.1.1 Smart TVs, Platforms, and App Stores

Smart TVs are basically TVs that can be connected to the internet. They come in two forms: (1) external dongles that plug into any TVs that have HDMI inputs (Apple TV); and/or (2) built directly into the TV (Roku TV). They have their own platform, such as Roku OS for Roku, Fire OS for Fire TV, and tvOS for Apple TV. In addition, smart TVs, like ones from Samsung, can rely on open-sourced platforms like Android. The platforms coincide with their own app stores that provide apps for various purposes such as video streaming, gaming, utility apps. Moreover, they provide advertising and tracking opportunities for app developers and the platform provider.

2.1.2 Fire TV App Store

Amazon's app store offers around 4K free apps during the time of our experiments (2019). Its app store is available through using the Fire TV device and a web version at `amazon.com`. We find a wide range of categories of apps — Fire TV is not just a streaming device, but can also be used for utility (*e.g.* calculators, alarm clocks) and games. By exploring the web version, we find that we can also install apps to any Fire TV device that is registered under our account. We use this knowledge later in Sec. 3.1 to scale the installation of apps across six different Fire TV devices.

2.1.3 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 use the fact that Fire OS is based off Android to explore whether VPN-based tools like AntMonitor [7] can be used for network traffic collection, described in Sec. 3.2.1.

2.1.4 Ads on Fire TV

As a first step to exploring ads on Fire TV, we looked at where ads are displayed on the device. Through navigating the Fire TV dashboard, we notice ads can be displayed on the dashboard itself in between list of featured apps. Next, when utilizing a sample of Fire TV

apps, we find ads in between streaming videos. Given these insights, we know that our automated tool, Firetastic, should capture network traffic that belongs to the platform (in addition to the traffic caused by apps) and trigger videos, so that ads can be played.

2.2 Related Work

2.2.1 Desktop, Mobile, and VR ATS Ecosystems

The desktop [21, 12, 10] and mobile [31, 33, 35] ATS ecosystems have been extensively studied. Englehardt et al. [10] crawled 1-million sites on desktop and utilized desktop-specific blocklists such as EasyList and EasyPrivacy [37] to identify ATS — Google, Facebook, Amazon, and Twitter, dominated tracking on the web. Razaghpanah et al. [31] explored Android apps and collected network measurements. They found that ATS are similar to ones on desktop and concluded that cross-tracking between desktop and mobile were widespread. Shuba et al. [35] looked beyond just ATS for Android by building decision tree classifiers to block ATS across apps. For the Oculus VR platform (owned by Facebook), Trimananda et al. [38] observed that the ATS ecosystem is in its infancy, with only a few major players for tracking like Unity and Google, while ad services were completely missing.

2.2.2 Smart TV ATS Ecosystem

The smart TV ATS ecosystem has not been extensively studied. However, concurrent with this work, three other works also studied smart TVs [32, 14, 24]. Ren et al. [32] investigated smart TVs and IoT devices. They concluded that these devices contacted third parties more than any other device, motivating our Fire TV study (and the published version [39]). Huang et al. [14] collected network traffic of smart TVs that were utilized by real users, as opposed

to testing apps in a testbed setting like our study. Most similar to our work, Moghaddam et al. [24] instrumented the Fire TV platform and studied ATS and PII exposures. Our work corroborate the findings of these works — that ATS is prevalent on Fire TV. However, [24] utilizes a heuristic approach to exploring apps on Fire TV by simulating predetermined sequences of button presses. Conversely, Firetastic explores apps using breadth-first search.

Chapter 3

Firetastic Tool

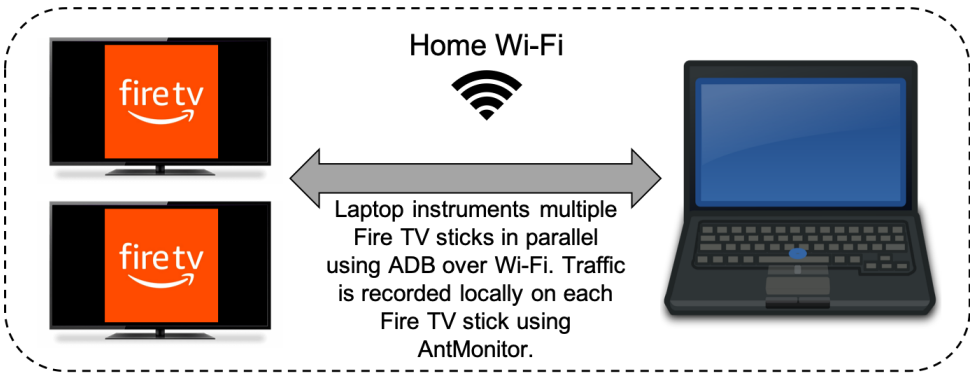


Figure 3.1: Firetastic: Installs Antmonitor [7] on each Fire TV device and uses one laptop to instrument multiple Fire TV devices. Uses Droidbot [19] to explore apps in a breadth-first search manner.

In this chapter, we describe the design and evaluation of Firetastic. In Sec. 3.1 we detail our app selection and extraction process. In Sec. 3.2, we explain how Firetastic can automatically explore and collect network traffic using AntMonitor and DroidBot. Lastly, in Sec. 3.3, we evaluate Firetastic against a baseline dataset through manually testing apps.

3.1 Selecting and Extracting Fire TV Apps

App Selection. To test the most relevant apps, we pick the top-1K apps from Amazon's curated list of "Top Featured" apps. We ignore 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. Our dataset covers approximately 25% of the total free apps (out of 4K).

Extracting Apps. Since we use a lower end model of Fire TV, the device cannot hold a large number of apps at once. As a result, we do data collection in batches of 50 apps at a time. First, we use the web version of the app store to install apps concurrently on six different devices; each device having a different set of 50 apps. We wait for the apps to download and install, then extract the apps using a python script. Once we have the apps, we move to our automated process of data collection, described in the next section.

3.2 Automated Data Collection

3.2.1 Network Traffic Collection with AntMonitor

Collecting Network Traffic. 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` [5]), 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, Antmonitor [34, 7], that does not require rooting the device . We discard incoming traffic because video content results in

huge PCAP files that slow down the experiments significantly, due to a technical limitation of ADB (*e.g.* slow transfer speeds for large files). The outgoing traffic is sufficient for analysis of the ATS ecosystem.

Modifications to AntMonitor. We make modifications to AntMonitor to have it work well on Fire TV. First, since AntMonitor outputs PCAPS that are each 10MB, we double it to 20MB to reduce the overhead of closing and opening a new PCAP during data collection for streaming video. This allows videos to stream with less interruption. Next, we make the setup of AntMonitor during the first launch more streamlined. For instance, on stock AntMonitor, the user would need to choose which apps AntMonitor would collect traffic for. Instead, we automatically choose all apps. Next, AntMonitor relies on the user to interact with the app to start the data collection. To have it work with Firetastic, we make it listen to a broadcast intent, so that Firetastic can start and stop the VPN automatically. Furthermore, AntMonitor hashes PII to obfuscate it. Here, we turn off this feature. We also make sure that TLS decryption is toggled on automatically. Lastly, we make minor UI changes such as changing TextViews into Buttons so that the user can navigate AntMonitor UI easily with the Fire TV remote.

3.2.2 App Exploration with Droidbot

In order to characterize the ATS ecosystem of Fire TV apps, we must explore (*i.e.* use) the app in a way that triggers network requests, especially ones that retrieve ad content. To scale it to thousands of apps, automatic exploration of apps is necessary.

App Exploration. To automatically explore each Fire TV application, we utilize Droidbot [19]. 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. For app exploration, we deduce that developers would minimize the necessary clicks in order to reach the

core content of their apps, especially for playing video content. Thus, we configure DroidBot to utilize its breadth-first search (BFS) 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-frst Search [DFS] algorithm may cause the automation to deep end into a path that we do not care about, such as an About view). With some trial and error, we select 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.

Modifications to Droidbot. We apply additional modifications to Droidbot to make it work on Fire TV devices. First, we make sure it launches the app by using the activity meant for a TV. To do so, Droidbot looks at the manifest file within the APK. Our modification causes Droidbot to choose the activity that contains the string "tv.". This ensures that the app is started as a TV app. Second, for apps that cannot be reinstalled on the device, we introduce a timeout parameter so that it does not wait forever during installation. Lastly, since the lower end Fire TV device that we use is slow when starting larger apps, we make sure Droidbot waits for 15 seconds after the launching the app.

3.2.3 Firetastic Algorithm

We summarize Firetastic's automation algorithm in Listing 3.1. For each app, Firetastic first starts the local VPN to capture (and decrypt) traffic. Next, it invokes DroidBot, which launches the app and explores it using BFS. When the 15-minute exploration completes, Firetastic stops the local VPN and extracts the `.pcapng` files that were generated during testing. Then, it uninstalls the app and continues to the next app. Testing one app at a time allows us to attribute the collected network traffic to the current app and the platform.

```
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 3.1: Algorithm for exercising FireTV apps.

3.3 Evaluation of Firetastic

We evaluate Firetastic in two ways. First, we look at whether it can automatically cause video to be displayed (*e.g.* for streaming apps). This tells us whether we are capturing video ad content during our data collection. Second, we look at whether it has good coverage of ATS domains (following the labeling methodology in Sec. 4.1.1). To do so, we rely on manual analysis of a sample of apps and treat it as a baseline against our Fire TV dataset.

		App Name	Playback? (A)	eSLDs A	eSLDs M	eSLDs $\frac{A}{M}$	ATS domains A	ATS domains M	ATS domains $\frac{A}{M}$
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 3.1: Video playback success for Firetastic, and a comparison of the number of domains discovered by 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 (A) and 16 minutes of manual (M) interaction. Not shown on this table: we were able to play video for all apps manually.

3.3.1 Baseline Network Traffic for Evaluation

To get a baseline of network traffic that corresponds to a real user, we run additional experiments for 15 apps. We select apps based on the top-10 apps and then randomly sampled five additional apps. For each app, we follow two different scenarios. For the first scenario, we use the app as a real user, this is denoted as the *manual interaction* (*i.e.* no automation). For consistency, we follow a protocol in which we attempt to play seven different videos for approximately two minutes each, leaving a few minutes to navigate between videos. For the second scenario, we run Firetastic while manually monitoring what is displayed on the TV screen, this is denoted as the *automated interaction*. For both scenarios, we run the experiment for approximately 16 minutes. Table 3.1 presents the overview of our results.

3.3.2 Can Firetastic Trigger Video Playback?

We find that Firetastic is able to trigger video playback for 10 out of the 15 apps (*i.e.* 67%), 60% for the top-10 apps and 80% for the random apps. Firetastic fails to trigger video playback for apps that have content that is locked and the free content is not easily accessible (*i.e.* the user needs to do more clicks to get to the free content). This is not surprising as Firetastic relies on a BFS approach to exploring the apps because it expects apps to have good UI design — minimizing the number of clicks necessary for users to get to the actual content. 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.

3.3.3 Can Firetastic Capture Domains through Exploration?

Firetastic's exploration approach is effective to collect the majority of the domain space for an app. We find that 10 out of 15 apps (67%), Firetastic uncovers 0.8 times the number of eSLDs, and 0.7 times the number of ATS domains when compared to the manual experiments. Interestingly, Fire TV collects more eSLDs for six apps and more ATS for seven apps, than the manual experiments. Importantly, this highlights the fact that video playback is not the sole trigger for eSLDs and ATS domains — usage of the app through other features, such as exploring all menus, can just be as worthwhile in discovering more domains. An improvement to Firetastic (as well as motivation for future tools) would be to make sure it balances between triggering video playback and exploring the other menus of the app.

Chapter 4

Fire TV's Ads and Tracking Ecosystem

In this chapter, we analyze our Fire TV dataset to characterize the ATS ecosystem. Sec. 4.1 gives an overview of our dataset and how we label domains as ATS. In Sec. 4.2, we detail how prevalent ATS is for the Fire TV platform. In Sec. 4.3, we compare it to other platforms: Roku, Android, and Oculus VR.

4.1 Fire TV Testbed Dataset

Number of	**Fire TV**
Apps exercised	1010
Fully qualified domain names (FQDN)	1734
FQDNs accessed by multiple apps	603
URL paths	240713

Table 4.1: Summary of the Fire TV testbed dataset.

The dataset collected using Firetastic are summarized in Table 4.1. For Fire TV, we discover 1734 distinct FQDNs, 603 of which are contacted by multiple apps. Firetastic uncovers

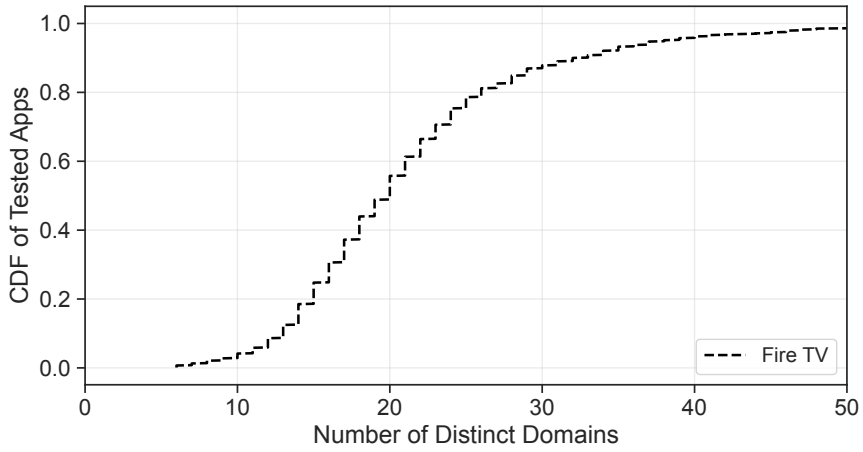


Figure 4.1: CDF of tested apps for distinct domains.

approximately twice as many FQDNs as [24], possibly due to longer experiments (*e.g.* 15 *vs.* 5 minutes, respectively) and different app exploration goals (*e.g.* exploring the views of each app *vs.* attempting to play video ads). Furthermore, Fig 4.1 reveals that the majority of apps contact many distinct FQDNs — 60% of apps contact between 15–25 distinct FQDNs.

4.1.1 Labeling the Dataset

To study the ads and tracking ecosystem on Fire TV, we focus on analyzing the domains that we collected during our data collection. As a result, we employ three different labeling approaches onto our dataset to identify ads and tracking domains, the parent organization that is associated with the domain, and whether the domain is first party, third party or platform-specific party w.r.t. the app that contacts it.

Ads and Tracking. For figures that denote top domains, we check if the FQDN is labeled as ATS by VirusTotal, McAfee, OpenDNS [40, 22, 26], or if it is blocked by *any* of the four blocklists considered in Sec. 5. For figures and tables that involve the entire dataset, we only utilize the blocklists for ATS labeling.

Parent Organization. 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 [25, 15], and use Crunchbase [8] to identify the parent company of the eSLD. For example, `youtube.com` belongs to Alphabet.

App-Level Party Categorization. To provide further context into the relationship between the app and the destination it contacts, we consider a destination as first party (*i.e.* the app or developer), third party, and platform-specific party (*i.e.* Amazon).

1. We first tokenize app identifiers and the eSLD of the contacted FQDN (we obtain the eSLD using Mozilla's Public Suffix List [25, 15]). For Fire TV, we tokenize the package names and developer names. For app/package tokens, we ignore common and platform-specific strings like "com", "firetv", 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*.

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 [34] ability to label each connection with the responsible process.

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.

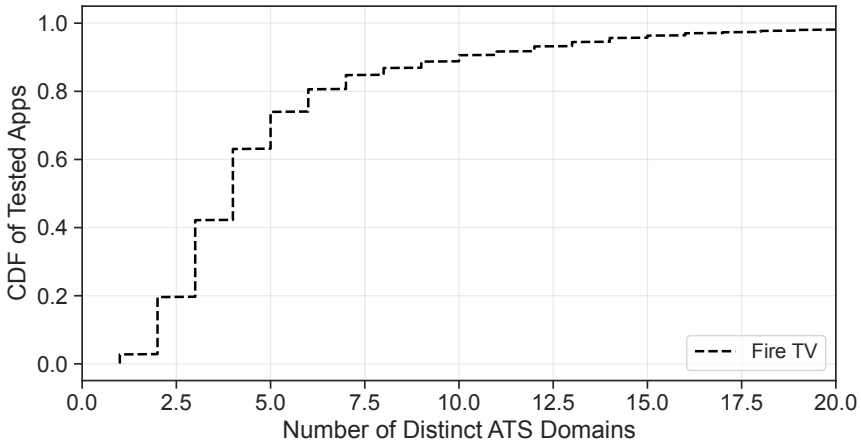
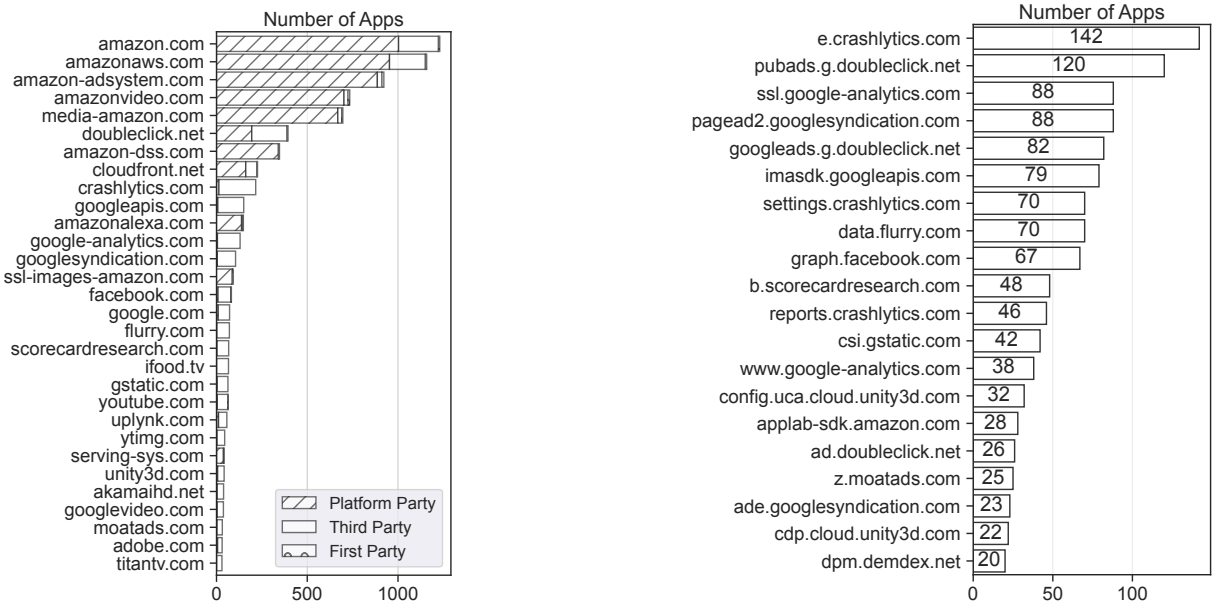


Figure 4.2: CDF of tested apps for distinct ATS domains.



(a) Top-30 eSLDs

(b) Top-20 third party ATS domains

Figure 4.3: (a) Top-30 eSLDs that are prevalent among multiple apps; (b) Top-20 third party ATS domains prevalent among multiple apps.

4.2 Fire TV ATS Ecosystem

Overview. To study the prevalence of ATS on Fire TV, we apply our labeling methodology, as described in Sec. 4.1.1 to our dataset and highlight insights from different vantage points. Overall, we find 512 distinct ATS FQDNs and 238 distinct eSLDs from our dataset of 1010

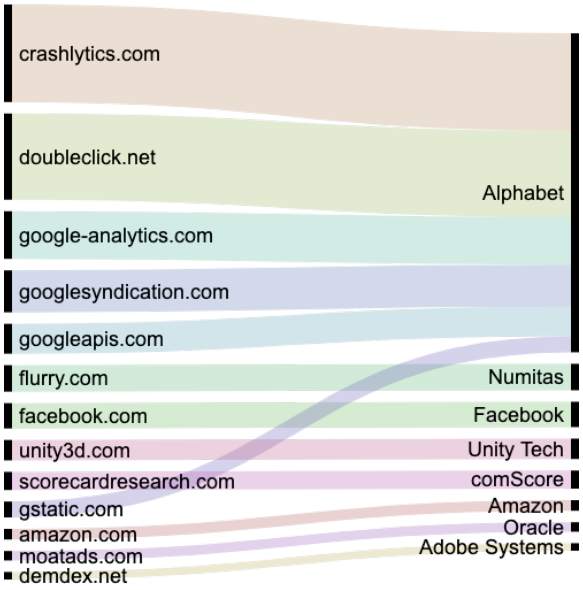


Figure 4.4: Parent org. of Top-20 third party ATS

apps — suggesting the prevalence of ads and tracking on Fire TV. Fig. 4.2 reveals that 60% of apps contact three to six distinct ATS domains, while 10% of apps contact 10–20 distinct ATS domains.

Prevalence by Number of Apps. We look at the prevalence of destinations based on the number of apps that contact it. In particular, Fig. 4.3a shows that Amazon, the platform provider, dominates the ads and tracking ecosystem on Fire TV — `amazon-adsystem.com` has close to 1K apps contacting it. Interestingly, we observe that first party eSLDs are rare and suggests that app developers rely on ATS libraries from third party and platform-specific party providers.

Next, we focus on third party ATS and their prevalence, as illustrated by Fig. 4.3b. Alphabet has a strong presence in the ATS space for Fire TV, with `*.doubleclick.net` and `pageads2.google-syndication` domains. Its analytic services, `google-analytics.com` and `crashlytics.com`, also rank high. Smaller ATS domains include `b.scorecardresearch.com` (tracking service) and `dpm.demdex.net` (audience data provider).

Parent Organizations. Fig. 4.4 shows the connections between third party ATS to their parent organizations. We use Crunchbase to determine this information in Sec. 4.1.1. Alphabet dominates the third party ATS ecosystem. Other notable organizations include Numitas from `flurry.com`, Unity Tech from `unity3d.com`, and Facebook from `facebook.com`

4.3 Fire TV ATS Ecosystem *vs.* Other Platforms

In this section, we compare the Fire TV ATS ecosystem with other platforms.

4.3.1 Compared to Roku

In the published version of this work [39], we also characterize Roku's ATS ecosystem. We find that both platform providers, Roku and Amazon, dominate their respective ATS ecosystem. When considering ATS based on eSLDs, we find an overlap of only 32%, suggesting that the ecosystems differ. When focusing on third party ATS organizations, Roku has some that are not prevalent on Fire TV, such as comScore, The Trade Desk, and RTL Group. Conversely, organizations found on Fire TV such as Unity Tech, Facebook, and Numitas, are not popular on Roku.

4.3.2 Compared to Android

We do a direct comparison of Fig. 4.3b to Razaghpanah et al. [31] findings on Android, focusing on third party only. Fire TV is similar to Android with an overlap of nine ATS FQDNs, seven of which are owned by Alphabet. This is expected, given that Fire TV is based off of Android and natively supports the ATS libraries of Android. Facebook (`graph.facebook.com`) and Verizon (`data.flurry.com`) both have a strong presence on

Fire TV and Android. Notably, third party ATS observed for Fire TV only include Adobe (`dpm.demdex.net`) and Amazon (`applab-sdk.amazon.com`).

4.3.3 Compared to Oculus VR

We do a direct comparison of Fig. 4.3 to Trimananda et al. [38] findings on Oculus VR. Both platforms dominate the ATS ecosystem — Amazon for Fire TV and Facebook for Oculus. For third party ATS, we observe that they share Unity (`unity3d.com`) and Alphabet (`google-analytics.com`). However, noticeably different is that Oculus does not contain ad related services like Fire TV (`doubleclick.net`, `moatads.com`). As noted by [38], this is not surprising because there are currently no on-device ads for the Oculus. Overall, Fire TV's ATS ecosystem is more diverse and mature than on Oculus.

Chapter 5

Effectiveness of Blocklists for Fire TV

In this chapter, we evaluate whether users can utilize DNS-based blocklists to preserve their privacy by blocking ATS. In Sec. 5.1, we explain what blocklists are and describe the four that we have chosen to evaluate. Sec. 5.2, we reveal the block rate of each blocklists when applied on our Fire TV dataset. In Sec. 5.3, we evaluate their effectiveness at blocking ads without breakage. Sec. 5.4 illustrates how much PII exposures are stopped by employing the them. Lastly, Sec. 5.5 recommends how to improve the blocklists for Fire TV.

5.1 What are DNS-based Blocklists?

In 2019, there are no viable commercial applications that can be installed on the Fire TV to block ads across all apps on the device. As a result, we use DNS-based blocking solutions, such as Pi-hole [28], that can monitor all DNS traffic from in-home devices and block them based on blocklists. Notably, this is a state-of-the-art solution that many users have deployed for free. Second, blocklists, lists that contain hostname syntax, are manually curated by domain experts or crowdsourced by users.

Specifically, Pi-hole acts as a DNS server and the user can configure their router to use this DNS server to monitor and block network traffic that matches blocklists. For instance, 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 [29].

For our work, we select popular DNS-based blocklists, while also considering their relevance to smart TVs. Specifically, we look at the following blocklists:

1. **Pi-hole Default** (PD): We test blocklists included in Pi-hole's default configuration [27] to imitate the experience of a typical Pi-hole user. This set has seven hosts files including Disconnect.me ads and 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" [41], to emulate the experience of an advanced Pi-hole user. This includes: Disconnect.me ads, hpHosts, a dedicated smart TV blocklist, 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 [23] 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 [17]. 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 [13]. SATV contains a total of about 3K entries.

Platform	# Domains	Block Rate (%) PD	TF	MoaAB	SATV
Fire TV	1734	22%	27%	22%	9%

Table 5.1: Block rates of the four blocklists when applied to the domains in our Fire TV dataset.

5.2 Block Rates

Table 5.1 shows the block rate of each blocklist across our dataset. We see that the the first three blocklists provide similar coverage, around 22–27% of FQDNs were blocked. Surprisingly, the smart TV specific block list "StopAd" had the lowest block rate. However, this does not mean that "StopAd" is not effective in terms of blocking ads. As a result, we look at whether the number of rules really matter in the next section.

5.3 Blocking Ads without Breakage

To evaluate whether blocklists are effective at blocking ads without causing functionality and visual breakage, we employ manual analysis.

Setup. We select 10 apps, six are top apps and three are randomly selected from our dataset. We then run five experiments per app, one experiment with no blocklist (*i.e.* No List), and the other four corresponds to each of our blocklists (applied using Pi-hole). 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 and visually observable missed ads. 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.

		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
Fire TV	Top	Pluto TV	✗	✓	✗	✓	✗	✓	✗	✗	✗	✖
		iFood.tv	✗	✓	✓	✓	—	✖	—	✖	✓	✓
		Tubi	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
		Downloader	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
		The CW for Fire TV	✗	✓	—	✖	—	✖	—	✖	✗	✓
		FoxNow	✗	✓	—	✖	—	✖	✗	✓	✗	✓
		Watch TNT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Random	KCRA3 Sacramento	✗	✓	✓	✓	—	✖	—	✖	✗	✗
		Watch the Weather Channel	✗	✓	✓	✓	✓	✓	✗	✓	✓	✓
		Jackpot Pokers by PokerStars	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗

Table 5.2: Missed ads and functionality breakage for different blocklists when employed during manual interaction with 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.

Results. Table 5.2 summarizes our results. We find that none of the blocklists were able to block all ads without causing some kind of breakage. PD was the most successful at blocking ads for seven out of the 10 apps (70%). However, it did cause two instances of major breakage. Notably, TF, which had the highest block rate of 27% (see Table 5.1), caused the most major breakage for four of the 10 apps (40%). Conversely, SATV, which had the lowest block rate of 9%, actually blocked more ads and caused only one major breakage, when compared to TF. Thus, the block rate may not be an indicator of the effectiveness of the blocklist, but rather whether the list was curated for smart TV devices.

PII	Fire TV Testbed Dataset (Apps & eSLDs)				
	1st Party	3rd Party	Platform Party	Other	Total
Advertising ID	17/7/25%	53/31/78%	715/4/71%	5/5/40%	725/39/71%
Serial Number	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	1/2/0%	2/2/100%	1/1/100%	-	4/5/40%
MAC	-	2/2/100%	-	-	2/2/100%
Location	-	27/7/90%	2/2/100%	-	28/7/90%

Table 5.3: 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. 4.1.1.

5.4 PII Exposures

We evaluate whether the four blocklists can adequately prevent PII exposures from Fire TV devices. We consider PII such as Advertising ID, Serial Number, and Location. If any of the four blocklists can block the exposure, then we consider that as blocked. Table 5.3 provides our results broken down by the number of applications receiving the PII, the number of eSLDs and the number of FQDNs that are blocked by any of the four blocklists. We further break down the PII exposures by first party, third party, and platform-specific party, following the labeling methodology of Sec. 4.1.1. This is necessary because it helps us infer the purpose of the exposure.

Extracting PII. To find PII, we use the Fire TV setting menus and the user account that we used to log into the device. Since trackers are known to encode or hash PII [9], 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 Sec. 3.2.1 that we can analyze HTTP information even for encrypted flows in the Fire TV dataset due to AntMonitor's TLS decryption [34, 7].

First Party PII Exposures. The majority of first party exposures are not blocked. This

Hostname	PD	TF	MoaAB	SATV
myhouseofads.firebaseio.com	✗	✗	✗	✗
mads.amazon.com	✗	✗	✗	✗
ads.aimitv.com.s3.amazonaws.com	✗	✗	✗	✗
analytics.mobitv.com	✗	✗	✗	✗
events.brightline.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 5.4: 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").

may be because they are necessary for functionality, personalization, or improving the application based on app usage. Interestingly, 17 apps receive the Advertising ID and only 25% of the FQDNs are blocked. We note that blocklists generally avoid blocking first party because it reduces the chance of breakage.

Third and Platform-specific Party PII Exposures. PII exposures to third party and platform-specific party domains can be for ATS purposes. Our blocklists capture the majority of PII exposures to third party domains, having high coverage for Username, MAC, Location and Advertising ID exposures. However, the most prevalent PII, Device ID is sent by 153 apps and only 36% of the FQDNs are blocked. For platform-specific party, we see high exposures of Advertising, Serial Number, and Device ID. 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". On the other hand, some exposures seem to serve a functional purpose. For instance, 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)" [6], 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.

5.5 Missed by Blocklists

Since there are not many smart TV specific blocklists, we leverage our Fire TV dataset to identify domains that should be blocked. First, a simple approach is to utilize PII exposures. For example, Table 5.3 reveals many FQDNs are still receiving the Advertising ID and are not blocked. A second approach is to use common keywords found from our dataset such as "ads", "track", and "metric". By relying on keywords, we are able to identify at least ten domains that are not captured by any of the four blocklists, as shown in Table 5.4.

Chapter 6

Conclusion

6.1 Summary

In this thesis, we provide one of the first large-scale studies of Amazon's Fire TV ATS ecosystem. To that end, we build Firetastic, a tool that can automatically explore and collect network traffic for Fire TV apps. Using Firetastic, we explore 1010 app, identify ATS using DNS-based blocklists and online URL classifying services. We observe that ads and tracking are prevalent on the Fire TV platform. First, we note that the platform provider (Amazon) dominates the ad ecosystem, as hundreds of apps contact Amazon related ATS. For third party, parent organizations such as Alphabet, Facebook, Numitas, are prevalent. Fire TV ATS ecosystem is similar to Android's but is more diverse when compared to Oculus VR. Furthermore, we find that DNS-based blocklists fail to block the majority of PII exposures for Device ID and Serial Number to third party destinations. Notably, the blocklists are not effective at blocking ads without causing some breakage.

This work was published in PETS 2020 [39]. In addition, we open source Firetastic [18] and make the dataset of 1010 apps available to the public. Since the publication of [39], it has

garnered attention from Consumer Reports and featured at FTC's PrivacyCon 2021 [30].

6.2 Future Directions

We can improve Firetastic to trigger more video ads by first using additional information like the app category (*e.g.* free streaming app, games) to select the appropriate exploration algorithm (*e.g.* BFS *vs.* DFS *vs.* others). Since we did not conduct studies using various Fire TV devices, another study for higher-end Fire TV devices and TVs with built-in Fire TV would serve as a good comparative analysis study.

We also have shown that there is a need to have curated blocklists for Fire TV to not only block ads but prevent breakage. Future work can look at how to automatically identify hostnames to block. Similar work has since been published in 2021 to identify non-essential traffic on IoT devices [20], which may serve as a guide to improve blocklists. Futhermore, future work can focus on improving the blocking capability for smart TVs. For example, use AntMonitor as a on-device app to block ads on Fire TV [35], which provides more granular blocking capabilities beyond hostnames (*e.g.* blocking based on URL paths, HTTP headers).

Bibliography

[1] Amazon VP shares how Fire TV is reimagining the largest screen in your home. https://www.aboutamazon.com/news/devices/amazon-vp-shares-how-fire-tv-is-reimagining-the-largest-screen-in-your-home, Sep 2021.

[2] Amazon Advertisers can now reach up to 50M monthly active users on Fire TV. https://advertising.amazon.com/blog/amazon-advertisers-can-now-reach-up-to-50m-monthly-active-users-on-fire-tv, Nov 2021.

[3] Amazon.com: Buyers Guide: Amazon Devices & Accessories. https://www.amazon.com/b?ie=UTF8&node=23477577011, Nov 2021.

[4] Amazon.com: TV with Fire TV. https://www.amazon.com/s?k=tv+with+fire+tv&i=amazon-devices&bbn=8521791011&rh=n%3A16333372011%2Cn%3A2102313011%2Cn%3A8521791011%2Cn%3A21579968011&dc, Nov 2021.

[5] Android tcpdump. https://www.androidtcpdump.com/.

[6] Antidot. Content Delivery Platform. https://www.antidot.net/content-delivery-platform/, 2019.

[7] AntMonitor open-source. https://github.com/UCI-Networking-Group/AntMonitor.

[8] Crunchbase. https://www.crunchbase.com/.

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

[10] S. Englehardt and A. 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.

[11] UCI Rokustic and Firetastic Dataset | UCI Networking Group. https://athinagroup.eng.uci.edu/projects/smarttv/data.

[12] P. Gill, V. Erramilli, A. Chaintreau, B. Krishnamurthy, K. Papagiannaki, and P. 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.

[13] Google LLC. apkanalyzer. `https://developer.android.com/studio/command-line/apkanalyzer`, 2019.

[14] D. Y. Huang, N. Apthorpe, G. Acar, F. Li, and N. Feamster. IoT Inspector: Crowdsourcing Labeled Network Traffic from Smart Home Devices at Scale, 2019.

[15] John Kurkowsi. tldextract. `https://github.com/john-kurkowski/tldextract`.

[16] 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].

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

[18] H. Le. Firetastic. `https://github.com/UCI-Networking-Group/firetastic`.

[19] Y. Li, Z. Yang, Y. Guo, and X. 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.

[20] A. M. Mandalari, D. J. Dubois, R. Kolcun, M. T. Paracha, H. Haddadi, and D. Choffnes. Blocking without Breaking: Identification and Mitigation of Non-Essential IoT Traffic. In *Proc. of the Privacy Enhancing Technologies Symposium (PETS)*, 2021.

[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] McAfee, LLC. Customer URL Ticketing System. `https://www.trustedsource.org/`.

[23] MoaAB: Mother of All Ad-Blocking. `https://forum.xda-developers.com/showthread.php?t=1916098`.

[24] H. Mohajeri Moghaddam, G. Acar, B. Burgess, A. Mathur, D. Y. Huang, N. Feamster, E. W. Felten, P. Mittal, and A. 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.

[25] Mozilla Foundation. Public Suffix List. `https://publicsuffix.org/`.

[26] OpenDNS Domain Tagging. `https://community.opendns.com/domaintagging/`.

[27] Customising Sources for Ad Lists. `https://github.com/pi-hole/pi-hole/wiki/Customising-Sources-for-Ad-Lists`.

[28] Pi-Hole: A black hole for Internet advertisements. https://pi-hole.net/.

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

[30] PrivacyCon 2021. https://www.ftc.gov/news-events/events-calendar/privacycon-2021.

[31] A. Razaghpanah, R. Nithyanand, N. Vallina-Rodriguez, S. Sundaresan, M. Allman, C. Kreibich, and P. Gill. Apps, Trackers, Privacy, and Regulators: A Global Study of the Mobile Tracking Ecosystem. *NDSS*, 2018.

[32] J. Ren, D. J. Dubois, D. Choffnes, A. M. Mandalari, R. Kolcun, and H. 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.

[33] J. Ren, A. Rao, M. Lindorfer, A. Legout, and D. Choffnes. ReCon: Revealing and Controlling PII Leaks in Mobile Network Traffic. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pages 361–374. ACM, 2016.

[34] A. Shuba, A. Le, E. Alimpertis, M. Gjoka, and A. Markopoulou. AntMonitor: A System for On-Device Mobile Network Monitoring and its Applications. *arXiv preprint arXiv:1611.04268*, 2016.

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

[36] G. 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.

[37] The EasyList Authors. EasyList. https://easylist.to/.

[38] R. Trimananda, H. Le, H. Cui, J. Tran Ho, A. Shuba, and A. Markopoulou. OVRseen: Auditing Network Traffic and Privacy Policies in Oculus VR. In *31st {USENIX} security symposium ({USENIX} security 22)*, 2022.

[39] J. Varmarken, H. Le, A. Shuba, A. Markopoulou, and Z. Shafiq. The TV is Smart and Full of Trackers: Measuring Smart TV Advertising and Tracking. *Proceedings on Privacy Enhancing Technologies*, 2020(2), 2020.

[40] VirusTotal. https://www.virustotal.com/.

[41] WaLLy3K. The Big Blocklist Collection. https://firebog.net.