uc3m | Universidad Carlos III de Madrid

Master Degree in Cybersecurity
Academic Year (e.g. 2020-2021)

Master Thesis

"ImposTer: Towards an extensible privacy analysis framework for Smart Home ecosystem"

Aniketh Girish

Tutor: Juan Manuel Estévez Tapiador
Advisors: Srdjan Matic & Narseo Vallina-Rodriguez
Madrid, 2021

ImposTer: Towards an extensible privacy analysis framework for Smart Home ecosystem

Aniketh Girish*
IMDEA Networks Institute / Universidad Carlos III de Madrid

Juan Tapiador
Universidad Carlos III de Madrid

Srdjan Matic
IMDEA Networks Institute

Narseo Vallina-Rodriguez
IMDEA Networks Institute / ICSI

ABSTRACT

The IoT ecosystem is an intricate network of stakeholders (*e.g.,* developers, advertising networks, platform operators) that often exploit data-driven business models. The ability of smart home platforms and devices to interact and exchange personal user data with each other opens potential privacy risks. But proposed black-box approaches to audit IoT platforms lack techniques to automatically and wholistically study their behaviour in rich execution environments where other devices and applications operate. This thesis proposes ImposTer, a cost-effective yet extensible privacy and security analysis framework for exhaustively studying the IoT ecosystem in conjunction with other devices. ImposTer utilizes a suite of analysis techniques to automatically capture and model the horizontal interaction across multiple devices in a consumer household. To this extent, along with multiple smart home devices, traffic interception techniques, and network protocol honeypots, I introduce a highly instrumented smart TV to empirically characterize inter-device communication. Using a dataset of 415 Smart TV applications, we expose information leakage and characterize interconnected nature of the smart home ecosystem, including the presence and interaction of potentially intrusive advertising and tracking services.

1 INTRODUCTION

The growing market of smart home technologies has evolved beyond basic domotic functionalities to systems such as personal assistants. Despite their benefits, IoT devices are susceptible to a myriad of privacy risks. With the integration of rich sensors such as GPS and microphone along with persistent Internet connectivity, these devices are powerhouses that could learn and expose sensitive information of not only about the environment but also about their users [17, 18, 20]. Internet-enabled Smart TVs are one such example which contains many features powered by sensors (*e.g.,* in-built camera) and tracking technologies inherited from smartphone platforms that would not be found on a conventional TVs. This technical development entails potential privacy and security risks [27]. Moreover, certain features of these devices can only be activated based on the data captured from the surrounding environment.

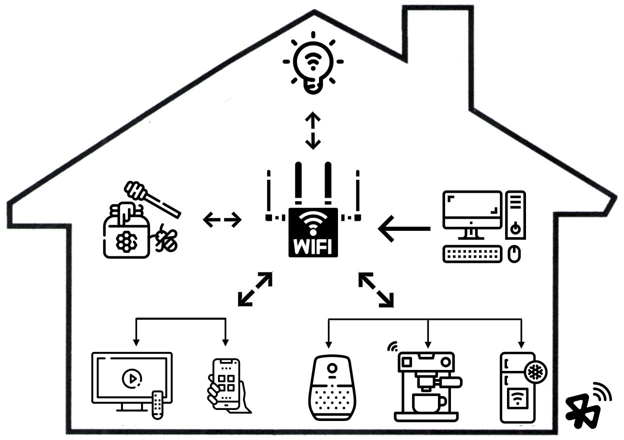

Figure 1: Our smart home emulated setup

Additionally, many of these devices can interact with products from different manufacturers. To encourage interoperable vendor compatibility, many IoT vendors build their devices based on prominent IoT platforms such as Android [21], SmartThings [29], or HomeKit [26]. These platforms enable IoT devices (even applications in these devices) to seamlessly communicate with neighbouring devices in a home network. However, prior research commonly abstracts IoT products as a self-contained monolithic entity that only interacts with the cloud, thus neglecting multitude of actors and dynamics present in the ecosystem. For example, it is a common practice in software engineering to integrate third-party services in order to speed up the development cycle. In several platforms like Android, iOS and most smart TVs, these libraries share the same privileges as the host application. Additionally, smart home devices are highly inter-connected systems capable of inter-device communication through platform-specific APIs. This potential hyper-connectivity can have negative consequences for users' privacy by enabling data

*Marked author is the lead student author of this paper.

leaks between devices to the cloud without user awareness. Unfortunately, manual inspection of IoT devices requires researchers to *hack* functionalities deep-rooted at the hardware level, limiting their scalability.

New regulations such as General Data Protection Regulation directive (GDPR) or California Consumer Privacy Act (CCPA) are meant to protect user's privacy, by mandating software products that collect personal data to portray a privacy policy in which they clearly explain data collection purposes, and prohibiting the collection of personal data without informed user consent. However, smart home consumer privacy are still at risk, as illustrated by the FTC settlement with Vizio for violating user privacy and collecting data without consent in 2017 [19]. Unfortunately, regulators lack tools for exhaustively and automatically auditing smart platforms. Prior work has reflected on the inter-dependencies that exist between smart platforms, and they leverage companion apps [59] or static analysis techniques [34, 41] to infer the privacy implication of IoT devices. Unfortunately, prior state of the art dynamic approaches falls short because they are either vendor [49, 58] or device-specific [38]. Consequently, the inter-connected nature of IoT devices (also within apps) and PII dissemination is largely understudied.

I propose a novel automated and extensible privacy analysis framework IMPOSTER for smart home infrastructures to enhance IoT platform transparency. The framework is composed by multiple elements that facilitate the discovery of potential privacy-intrusive behaviours across multiple platforms when combined with static and dynamic analysis techniques. The framework component includes 1) a static analysis provides insights regarding various third parties involved, resources requested by an app, code signatures, unique URLs and usage of network interfaces. 2) Dynamic analysis helps gathering insights during runtime and monitor network activity. 3) In addition, this framework includes a network honeypot that automatically collects and models behavioural patterns of IoT devices and applications. Furthermore, proprietary IoT devices such as popular smart home assistants and smart speakers are also introduced in my framework to passively capture their interactions. 4) Additionally, I introduce a heavily instrumented Smart TV to the framework, capable of monitoring apps' behaviour at the system and network level, including a proxy for intercepting encrypted traffic.

To evaluate my framework, I collect an app corpus of 415 TV apps by crawling the official Google Play Store. Then, I automatically interact with these apps on the Smart TV to simulate human behaviour through a "smart" UI Exerciser developed in-house. IMPOSTER extends the state of the art with my vendor-agnostic framework to characterize the entangled IoT infrastructure and the platforms' interconnected nature to audit privacy compliance by combining various monitoring techniques.

With these instrumentation, my main findings are the following:

- Application of IMPOSTER on smart TV apps and embedded third-party libraries reveals PII dissemination through trackable information such as Device ID or serial number.
- Frequent communication in the form of device scans has been observed among the incorporated IoT devices.

The framework - IMPOSTER presented in this thesis is a preliminary working prototype of a new approach to model IoT behaviors and their risks empirically. The framework will extend during my PhD thesis under the supervision of Dr. Narseo Vallina, as discussed in Section 6.1.

2 BACKGROUND

This section describes the most prominent Smart Home and Smart TV platforms. I describe prior research efforts that focused on studying their privacy and security risks empirically.

2.1 Smart home systems

Figure 1 represents a common smart home set-up, including TV streaming services, smart assistants, thermostats and light bulbs. A unique feature of these devices is their capability to be controlled by other devices in the environment (*e.g.*, voice commands from smart assistants). Commercially available smart home assistants act as a hub such as Smart Things [29], HomePod [26], providing a centralized control over devices from different vendors.

Home assistant Amazon and Google are two of the most well-known competitors in the market for virtual assistants. For example, Amazon Alexa and Google Home are non-headed devices without conventional I/O interfaces (such as a screen or a USB port). To interact with these virtual personal assistant devices, users use wake-words such as "*Alexa*" or "*Hey Google*". Notably, home assistants enrich their capabilities by introducing *skill* by Amazon or *action* by Google which are like apps but for these platforms. Despite the convenient features, there is an increasing concern of privacy and security risks involving phishing attacks or PII dissemination [33, 36–38, 45, 55]. More recently, Giese *et al.* [42] studies Amazon Echo Dot in isolation to reveal that, Echo dot retain private information such as user profile, WiFi credentials and much more even after a factory reset.

Smart Hubs enable connected devices to communicate with their companion app, the cloud back-end and even other devices. Samsung SmartThings [29], Google Nest [12], Apple HomeKit [26] are some of the smart hubs available in the market today. Using the companion apps users can interact with the hub, and perform actions such as installing third-party apps. Smart hubs have also received attention from researchers in the past for their privacy intrusiveness. Recently, Babun *et al.* introduces IoTWatcH [34] a dynamic analysis tool for exposing privacy violations in Samsung's SmartThings IoT apps. Similarly, Fernandes *et al.* [41] discovers security flaws of Samsung SmartThings IoT ecosystem by studying 499 apps.

2.2 Smart TV ecosystem

Smart TVs are one of the most popular smart home device. It allows consumers to stream content online, browse the Internet and even play games. In today's consumer market, Android based smart TV platform dominate with several popular vendors such as Sony, NVIDIA and Philips. Additionally, vendors such as Amazon introduce smart TVs with their own modified version of Android TV, the Fire TV. Due to the similarities within the Android platform for phones and the TVs, consumers get to install apps for their smart TV. However, the distribution of these apps is diverse with custom store operated by stakeholders such as Amazon. This third-party distribution model raises potential privacy risks (*e.g.,* distribution of potentially harmful apps) due to the limited vetting capabilities. Therefore, many researchers have explored the privacy compliance of these devices. Both Varmarken *et al.* [58] and Moghaddam *et al.* [49] explores the tracking and Advertising ecosystem on both Amazon Fire TV [2] and Roku TV [15] to find that a large number of TV apps expose PII to third party domains. Interestingly, Moghaddam *et al.*, reveals Smart TVs leak user viewing habit based on through their *best-effort* TLS interception.

Android TV apps Android based Smart TV apps are similar to mobile apps with some minor additional parameters set in the APK manifest by the developer to run on a smart TV. The apps for TV must declare a specific TV launcher activity. In contrast to mobile apps, TV apps must not declare unsupported hardware such as a touchscreen in the manifest. Finally, Android TV apps can opt to add support for **Leanback** library [11]. Leanback library is a part of Android development kit which provides user interface (UI) templates and navigation support that are exclusively for Android TV.

2.3 Holistic approaches

Many smart home devices implement APIs to enable integration with other software and platforms over the network. For instance, current platforms such as iOS and Android allow devices to be connected at all times. Apple iOS allows users to scan for nearby device with *Find my* app [6]. *Find my* app crowdsources location data from nearby devices without requiring any network connectivity. Similarly, Apple's new Airtags [5] uses similar technology to crowdsourced location data based on ARP scans. These techniques are not limited to smartphones or location tracking gadgets such as Airtags. Apple provides similar functionalities for the app running in their smart TV OS to scan for nearby devices [24, 28]. Analogously, Spotify Connect [30] feature allows the Spotify app to browse for nearby devices. Google Chromecast devices also scan for nearby compatible devices to initiate a pairing session [31].

Prior research has been successful in revealing security and privacy loopholes within the IoT ecosystem. Prior tools proposed by researchers only explore certain vantage points (*e.g.,* companion apps or network traffic) to answer IoT device behaviour [38, 54, 55, 57, 59]. To perform a complete characterization of smart home environment while opening new avenues for inferring platform and device behaviours, we require to take in account various components (*i.e.,* devices, platform APIs, companion apps *etc.*) collectively within the system - not just one. Therefore, in comparison to the state of the art, my work introduces a unified vendor agnostic methodology utilizing both the device and companion apps capable of characterizing smart home behaviours. The methodologies exposed by my framework ImposTer could be utilized to understand data flow and private data dissemination within the IoT black box as discussed next.

3 METHODOLOGY

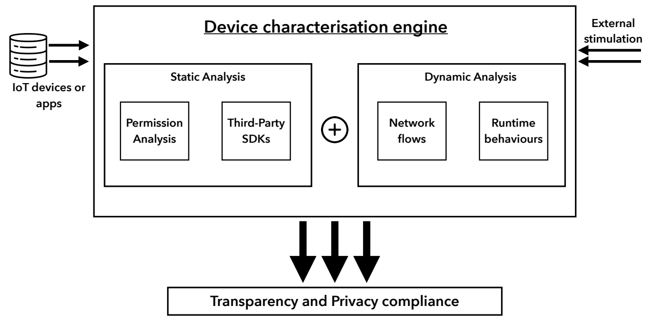

Figure 2: ImposTer Architecture, highlighting system components and the hybrid analysis approach responsible for auditing privacy compliance of IoT devices

ImposTer is a new framework designed to characterize the smart home IoT ecosystem to determine potential privacy-intrusive and harmful behaviours as illustrated in Figure 2.

Static and dynamic analysis methods present limitations when used separately [53]. Static analysis signals are often prone to false positives and can fail due to obfuscation methods. Additionally, applicability of static analysis is limited in case of certain IoT devices and platforms. For instance, all Amazon Alexa skills are hosted in the cloud and the code is dynamically loaded [39]. Static analysis techniques rely on studying the software's binary code, and thus have no visibility into dynamically loaded code. Moreover, the use of static analysis alone generally fails to contextualize the results into meaningful behaviours *e.g.,* understanding a potential privacy violation. Although dynamic analysis allows us to audit the runtime behaviour of an IoT device, IoT research with dynamic analysis techniques heavily rely on *hacky* hardware based black-box testing [42] and therefore, is not scalable for understanding and characterizing smart home devices at scale. In the light of these limitations, I explore new techniques that combine both existing static and dynamic analysis tools to extract a wider and more complete set of behaviors. I extended the tools with new approaches as discussed in the following section to increase their coverage, scalability, and visibility of the analyses.

3.1 Static preprocessor

I perform static analysis on Android based IoT apps, to gather signals that can be retrieved without having to execute it. I extract essential characteristics such as manifest information, unique strings (including URLs), code-signatures and network interfaces which are immune to obfuscation methods.

Permission Analysis Using static analysis, I investigate the use of Android app permissions to study what type of resources and data smart home apps request access to. In cases where sensitive resources are requested by an Android app, I study whether any such data is sent over the Internet or not. Using my suite of custom tools, I parse the Android Manifest [4] to collect invasive and dangerous permissions (*e.g.,* User location) requested by an app.

Third-Party services and libraries Also common know as Third-party SDK in mobile platforms, share the same privileges as of their host app, as discussed previously. Thus, SDKs are also capable of collecting and transmitting sensitive data. I identify third-party libraries embedded in an app using a suite of pre-existing static analysis tools such as LibRadar [48] and Exodus list [10]. Powered together with custom scripts based on Androguard [3], dexdump [9], and Baksmali [7] to parse an APK to collect information such as unique URLs or code-signatures, my static analysis pipeline is capable of extracting third-party SDK information of IoT apps without running them. I leverage LibRadar's classification data to shed some light on the relevant stakeholder involved in the smart home ecosystem as well.

3.2 Dynamic Analysis

When possible, ImposTer is capable of analysing dynamically the behaviour of any given IoT device introduced to the environment. However, effective dynamic analysis require granular instrumentation framework to facilitate the automated execution with external simulation to produce interesting behaviours. Since IoT devices are generally closed and proprietary in nature, building a scalable framework capable of instrumenting these devices is challenging. To overcome this limitation, ImposTer complements runtime execution with network-level monitors to infer potential devices and behaviors of other devices hosted in the same network.

3.3 Runtime Analysis

To facilitate runtime level monitoring, ImposTer supports low-level instrumentation of two smart TV platforms.

(1) Instrumented Android TV I build a universal Android TV on a Raspberry Pi 4 which helps to carry out experiments on apps meant for the TVs and model a universal behaviour of Smart TV apps. I implemented the port of Android TV to Raspberry Pi based on a pre-existing [23] work. After successfully porting Android TV for Raspberry Pi, I modified Android Open Source Project (AOSP) that corresponds to Android 11 for TV to monitor app execution based on prior research [53, 60, 61]. My instrumentation also allows us to create interactions and model behaviours among IoT devices within the home network. Which is otherwise, impossible to observe without instrumentation capabilites. The instrumentation allows us to comprehensively monitor the behaviour of each app at the Android-framework and network traffic levels.

(2) Amazon Fire TV I introduce a Fire TV to our framework. Fire TV by default runs a modified and proprietary version of Android. However, due to the customization added by Amazon on Fire OS, common black-box testing techniques would not cover the complete device behaviour. Additionally, since Fire OS is proprietary, source-code instrumentation is not possible. Therefore to enable instrumentation, I rely on Frida [25], a well-known dynamic code instrumentation toolkit to perform dynamic analysis. I implemented custom scripts using Frida which enables the collection of detailed runtime and network logs by monitoring Android API for any smart Fire TV app. My approach hooks into various networking and runtime modules using Frida without interfering with the operations of the host system and other co-located apps. However, to enable Frida, an Android device must be rooted. Fire TV does not facilitate rooting capablities. However, it is still possible to hook via Frida, but has it limitation while testing apps at scale. In addition to errors that my instrumentation could have introduced, it is also well known that hooking more than a few hundred functions using Frida causes the system to be unstable, resulting in frequent crashes [51]. Anyhow, regardless of the obfuscation techniques or anti-testing methods [47] apps use to disrupt static analysis, no app can avoid the instrumentation implemented on both the TVs, since it executes in the system space of the Android framework.

Companion apps Most IoT devices come with a companion app as their counterpart. They are mobile apps meant for interacting with the IoT device. In my case, to interact with non-headed IoT devices I make use of their companion apps. I run the companion Android apps on Google Pixel 3a running a highly instrumented Android 9, provided by AppCensus for research purposes [1]. This allows to capture smart home device and platform behaviour from their companion app point of view. Thus increasing the visibility of my framework to advocate better for privacy leaks. The instrumentation include comprehensive monitoring hooks at kernel, Android-framework, and network traffic levels capable of intercepting traffic and public API usage and calls. I manually, install, setup and interact with the IoT devices in the dataset using the companion apps.

Table 1: Prominent protocols implemented in the honeypot

Protocol	Port
UPnP	1900, 5000
HTTP	80
HTTPS	443
mDNS	5353
FTP	21
SMB	139, 445
Telnet	23
IPP	631

3.4 Network-level Monitoring

I facilitate network monitoring from various vantage points to receive a birds-eye view of the device behaviour. My framework emulates a regular home network environment using a custom access point build with a Raspberry Pi 4. IoT devices are connected to this access point and is placed behind a NAT setup. To intercept all network traffic, I deployed a transparent proxy server using mitmproxy [13] to record plain-text network flows. However, IoT devices are typically closed and proprietary, making it difficult to intercept encrypted traffic because we cannot install root certificates in order to monitor HTTPS traffic. In such cases, I decide to fall back to capture only encrypted traffic. Whereas, for instrumentable devices such as my custom Android TV, I installed mitmproxy's certificate into the system store to record plain-text traffic. Furthermore, to observe the network transmissions from the custom Android TV I build, I instrumented the relevant Java network libraries in AOSP (discussed in-detail in the following section) to improve visibility of the encrypted traffic and to have granular information of the network-level behaviour.

Honeypot Traditionally, honeypots have been used as decoys to mimic real devices on a network and help researchers understand the threat dynamics [44]. Ordinarily honeypots are used to study attacks, but in my case, I used the honeypot to opportunistically capture inter-device IoT communication and characterize privacy risks associated with IoT platforms by emulating any smart home device. Protocols implemented within the honeypot will stimulate unknown behaviours, which otherwise are missed out. Therefore, my honeypot implementation provides a high spectrum of visibility towards security and privacy risks of the IoT platforms. Additionally, to understand the aforementioned platform behaviours, I passively analyze the logs my honeypot generated based on the traffic captured to further automatically adapt the interactive-ness of my honeypot. I statistically analyzed the open data published by Alrawi *et al.* [50], to carefully select the protocols that are implemented in my honeypot as illustrated in Table 1.

3.5 Automatic App execution

Since my framework focuses on dynamic instrumentation, apps must be executed to monitor their behaviours. To scale testing beyond hundreds of app, manual interaction with these apps can be tedious. Therefore, to automatically simulate user-behaviour for apps and to scale app testing in my framework, I make use of Droidbot [46] - UI test input generator for Android. However, Droidbot is designed to work with Android devices and it doesn't work well straight of the box with Android TVs. Droidbot requires to parse the manifest file to interact with the app and the manifest files are different for Android TV apps. I tailor Droidbot for Android TV with additional instrumentation to various parsers within Droidbot (*i.e.,* logs, manifest *etc.*). In my setup, I configure droidbot to explore each app for 3 minutes and use its breadth first search algorithm to improve the probability of triggering a sensitive behaviour during the app exploration phase. My framework installs and explores each app separately (*i.e.,* installs an app, explores the app, capture logs, and finally uninstalls the app). With this flow, I make sure that there is no background app operations to minimize the contamination from other apps. Once the exploration completes, the custom tool build around Droidbot collects logs (includes both network and runtime logs) from the instrumented Android devices and is stored for later analysis.

3.6 Summary

Bringing all the aforementioned components together constitutes the first version of my framework. All the smart home devices available in my dataset (as described in section 4.1) is connected to the access point where both encrypted and unencrypted traffic is collected opportunistically. In-order to empirically measure the capabilities of the proposed framework, the instrumented Android TV executes apps based on the heuristics discussed in the automatic app execution section and the traffic is collected at the access point as well. Finally, the honeypot connected to the access point sneakily sits between the devices to capture smart home device behaviour in a rich execution environment.

4 HARDWARE & DATASET

In this section, I provide a detailed overview of the devices and IoT apps used for the evaluation of ImposTer framework.

4.1 Device dataset

In order to perform a realistic evaluation with ImposTer, I selectively curated the list of smart home devices based on the search results from e-commerce website such as Amazon to find most popular IoT devices and selected the device based on customer ratings. To capture device behaviours, my framework comprises of 6 different IoT devices and platforms. Table 2 describes the devices in my framework by category. In the test-bed, in addition to the devices, my framework contains, a Debian laptop, 2 Android phones and 1 IPhone. The mobile phones connected to the framework are used to control and interact with the IoT devices using their companion apps while the laptop is used to control the whole environment and automatically test smart TV apps.

Table 2: Categorized IoT devices in my dataset

Categories	Device	Platform
Smart TV	Amazon Fire TV	Fire OS
	Instrumented Android TV	Android
Home Assistant	Samsung SmartThings hub	SmartThings
	Amazon Echo dot	Alexa
	Google Nest hub	Nest
	Apple Homepod	Homekit
	Starling hub	-
Home Appliance	Amazon Smart plug	Alexa

4.2 App collection

I implemented a crawler to collect Android TV APKs from Google Play store. Unfortunately, crawling TV apps at scale from the playstore is not straight forward since they do not provide platform-specific (*e.g.*, TV, wear *etc.*) metadata. Therefore, my crawler took a heuristic approach to detect potential TV apps in the playstore. Once the APK is downloaded, I verify if it is meant for a smart TV by parsing the manifest to confirm the following: (1) It must publish a launcher TV activity. (2) The APK must not broadcast unsupported hardware such as touchscreen. (3) It can optionally add support for the Leanback library as discussed in section 2.

Despite the fact that my dataset comprises 415 applications for smart TV, I was unable to perform dynamic analysis on 5% of the apps since at least one of them failed to install or crashed the smart TV when started. A subset of these apps are popular apps with 1,000,000+ downloads in the playstore.

5 EMPIRICAL RESULTS

While the current implementation ImposTer is still incomplete, this section provides an overview of its potential to study a wide range of IoT privacy risks automatically. I empirically evaluate the effectiveness of ImposTer, using the instrumented smart TV prototype with Android TV apps to explore the following behaviors: (1) third-party SDKs; and (2) access and transmissions of sensitive data across devices and the cloud. The instrumentation and setup to accomplish the evaluation has been discussed in section 3.

5.1 Third-party components

As discussed previously, developers tend to heavily rely on third-party services (commonly known as SDKs) to speed up their development process. Advertising and Tracking services (ATS) are one of the most popular third party SDK in Android ecosystem. Android developers rely on such SDKs for monetization purposes [32].

Table 3 shows the Top-20 third party services found in my Android TV app dataset. Analytical services such as

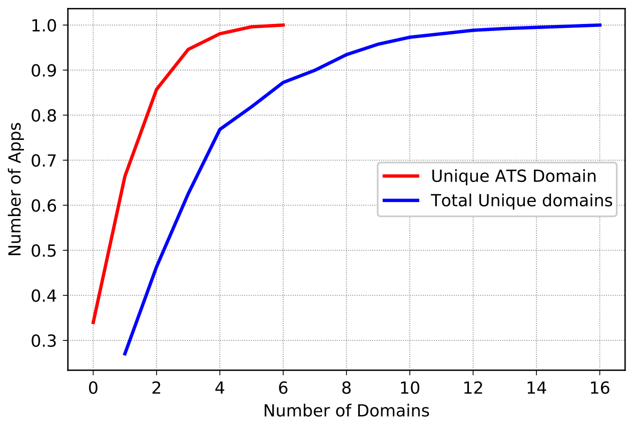

Figure 3: Distribution of Advertising and tracking domains with benign domains across smart TV apps

Table 3: Comparison of top-20 third-party SDK and ATS in apps from Libradar and Exodus respectively

Libradar			Exodus	
SDK name	Category	% of apps	SDK name	% of apps
Android Support v4	Development Aid	78%	Firebase Analytics	59%
Google Mobile Services	Development Aid	53.5%	CrashLytics	51%
Glide	Development Aid	33%	Admob	46%
Android Support v7	Development Aid	24.25%	Google Analytics	24.25%
Facebook	Social Network	22.75%	Facebook Login	20%
Google Gson	Development Aid	17%	Facebook Analytics	19.5%
Amazon In-App Purchasing	Payment	16.5%	Google Tag Manager	19.25%
Bolts Base Library	Development Aid	16.5%	Facebook Share	18.75%
Google Play	App Market	11.25%	Facebook Places	11.25%
OKHttp3.0	Development Aid	10.75%	Demdex	10.75%
Firebase	Development Aid	10.25%	AppsFlyer	10.25%
Fasterxml	Utility	9%	ComScore	8.75%
Crashlytics	Mobile Analytics	8.5%	Nielsen	8.5%
Amazon Auth SDK	Development Aid	8.25%	Conviva	7.75%
Apache Common	Development Aid	8%	Branch	7.5%
Amazon AWS	Development Aid	7.75%	Braze / Appboy	7%
Google Ads	Advertisement	6.5%	Moat	7%
Github	Development Aid	5.25%	IAB Open Measurement	6.75%
ExoPlayer	Development Aid	5.25%	Segment	6%
butterknife UI Framework	GUI Component	4.75%	Unity3d Ads	5.75%

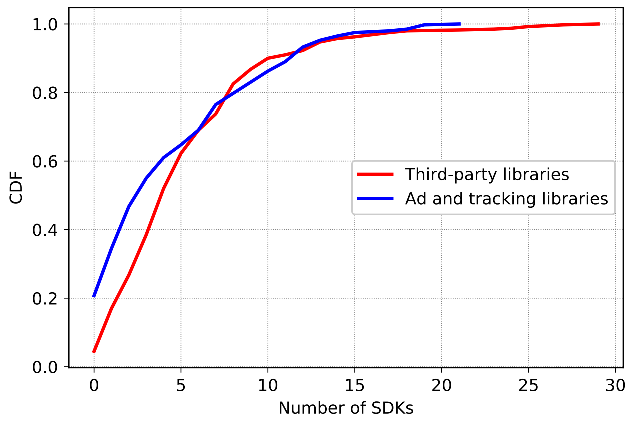

Figure 4: Distribution of SDKs across smart TV apps

Firebase, Admob, Analytics owned by Google predominantly exists at very large in the majority of the smart TV apps. In addition to the ATS SDKs in the table, my dataset contained ad libraries such as Tapjoy and Vungle. These are example ad libraries who specialize in connected smart TV services. Figure 4 illustrates the distribution of the SDKs in my dataset (including ATS services). In my dataset, I find atleast 60% of the app includes 5 ATS SDK, whereas in my dataset I found 117 unique ATS SDKs based on the Exodus results. Moreover, to identify the distribution of Advertising and Tracking SDKs across my dataset, I randomly hand-picked a list of 4 popular mobile SDKs from the study conducted by Razaghpanah *et al.* [32] to find that 35% of the TV apps in my dataset had atleast 1 of these SDKs embedded in the app.

Executing our app dataset on the instrumented smart TV generated 20K HTTP/S flows. Figure 3 shows the distribution of unique domains (including ATS domains) contacted by each app in my dataset. I used the data from the study carried out by Razaghpanah *et al.* [32] to filter ATS domains in my dataset to find that, there is atleast 10% of the apps that frequently contact and unique ATS domains in my dataset. Analyzing the traffic captured at both the network proxy as well as from the instrumentation, I found a few interesting cases:

- Obfuscated IP check: I found services, checking if the IP address is being obfuscated by the user. This could have legitimate purposes for example, fraud prevention. However, an intrusive intend could be Geo-location based tracking.
- isRooted & isDebuggable flags: These checks could be utilized to model app behaviour during runtime in-order to avoid scrutiny from security and privacy researchers.
- Browser info (model, version, type): Could be utilized for fingerprinting purposes.

I acknowledge that, these parameters seen in the traffic could have legitimate reasons for their existence. However, existence of these parameters in the traffic for smart TV at homes is concerning, because many of the smart TVs in the market are not instrumentable and lack debugging capabilities. This however, invites our attention to model developer behaviour in new smart home platforms. Smart home devices contacting third-parties be by itself should raise concern about the privacy implication of such behaviours.

5.2 PII dissemination

In this particular analysis, "PII dissemination" is the transmission of any Personal Identifiable Information from the smart TV device to any cloud endpoint. With static analysis I extract an upper bound approximation of potential access to permission-protected data. In Android ecosystem, sensitive system resources that can potentially be a data source for PII leaks are referred as dangerous permissions to denote protection levels.

As depicted in Figure 5, the most requested dangerous permissions in my dataset are related to storage, microphone and location. Additionally, many of the permissions requested by apps are not available in Android TV. This in-turn speaks about the developer landscape of Android TV apps. Developers develop cross-platform apps without considering the privacy implication of such development process. Not only this introduces over-privileged [35] apps in Android TV ecosystem, but also translates the advertising and tracking ecosystem to TV ecosystem. However, The high request rate of dangerous permission does not imply that smart TV apps are actually exposing sensitive data over the Internet. Therefore, I compliment my static analysis findings with dynamic analysis techniques as discussed in section 3.

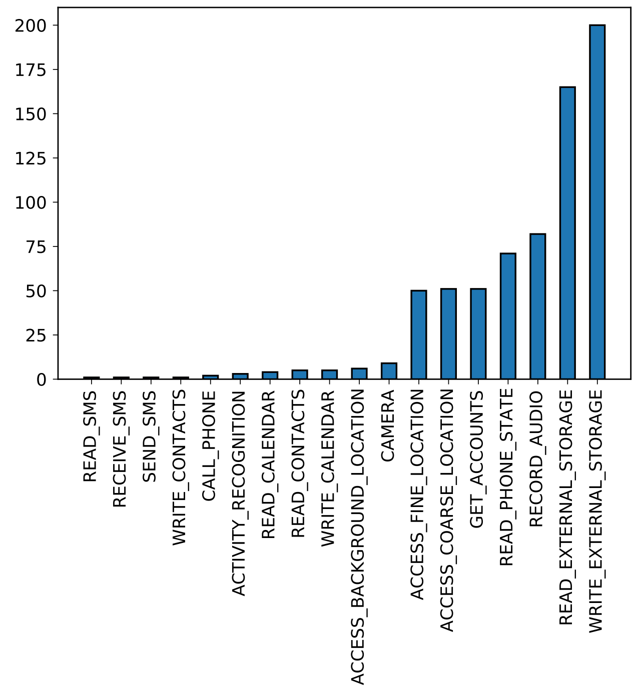

Figure 5: # of TV apps requesting dangerous permissions

Henceforth, my network-level instrumentation enables identifying PII values (such as advertising ID and serial number) through monitoring network traffic and analyzing the logs there after where I search for common PII values in the URIs. However, the number of PII exposure my analysis estimates a lower bound of all the possible instances of personal data dissemination that might transpire in my smart TV dataset. Because I cannot simulate every possible code-path even when using a smart monkey and because I cannot model every possible communication from other devices.

Table 4: Different type of personal information dissemination detected in logs generated by my smart TV instrumentation

Identifier type	Description
Android ID	Persistent ID created at Android setup
Serial number	Persistent hardware/manufacturer ID
Device ID	Persistent device ID
Advertising ID	Resettable Advertising ID
Router SSID	SSID of the access point device connected to
Country	Persistent user location until device is moved

As portrayed in Table 4, my framework detect transmission of identifiers such as Android ID, Device ID *etc.*. Inspite, Google's recommends developers to use the Android Advertisement ID (AAID) as the only user identifier [8] since it is resettable. Despite such regulations, I found that 62 (15.5%) apps shared non-resettable Device ID over the Internet. Whereas 7 apps were found combining AAID with Android ID within their request in-order to keep a persistent user profile. However, combining the AAID with other persistent identifiers such as serial number without explicit consent from the user has inverse effects and is in violation of Google's Terms of Service [16].

The network-level instrumentation instigated the discovery of these PII leaks within net-flows after intercepting them through MITM techniques. However, monitoring PII leaks of IoT devices which denies any instrumentation possibilities is challenging since they converse through encrypted channels.

6 DISCUSSION

For this study, I introduced 7 IoT devices to simulate a smart home environment and to monitor their behaviour. Throughout this study, these devices where sitting idle while connected to the Internet through my intercepting access point.

My ImposTer prototype shows the potential of this approach to study smart devices. It opens new opportunities to assess the limits of black-box testing on IoT products and the possibility of creating new types of IoT fuzzing methods. My analysis shows potential privacy violations in the form of persistent identifiers sent to third parties. Using the honeypot, my framework ImposTer has found evidence of cross-device scanning among devices in the network. Analyzing the traffic captured from both the honeypot and as well as from the access point reveals that, the devices in my test-bed has been predominantly scanning the network using various protocols and techniques. Samsung SmartThings and Google Nest uses ARP scans to identify nearby devices. Occasionally, I saw HTTP requests send from the Nest to my honeypot. Similarly, Amazon echo scans and discovers devices using UPnP protocol. However, scanning nearby devices in a smart home ecosystem might have an adverse privacy effect as it can be used to geo-locate users trough their WiFi AP but also to infer the socio-economic level of the household (and to discover vulnerabilities) by fingerprinting the types of devices that they have deployed. However, I note that these behaviour which ImposTer monitored may not necessarily be of malicious or privacy-intrusive intents if they are executed with informed user consent or for legit reasons needed to offer the intended service.

6.1 Limitation and future work

Static and dynamic analysis challenges For the empirical measurement conducted to evaluate my framework, I employ both static and dynamic analysis techniques. I assessed declared permissions, third-party libraries (including Advertising and tracking libraries) and data sharing over the Internet using this hybrid mechanism. However, I acknowledge the limitation of the aforementioned techniques, as they cannot guarantee a complete coverage and visibility of an apps behaviour (code or the data flows):

- First, static analysis methods have high false positive and false negative rates due to many factors such as code obfuscation, dead code, *etc*.. Hence, I consider the results obtained by static analysis as an upper-bound estimation of the potential harmful. Therefore, as discussed in the previous sections, I compliment my static results with dynamic techniques in-order to reduce the error rates.
- Second, both LibRadar and exodus use a pre-loaded database to identify third-party libraries and trackers. Therefore, this approach cannot detect new services that are not present in this database.
- Third, my dynamic analysis visibility is not complete. I need to incorporate additional instrumentation to other Android APIs such as other network libraries within AOSP to get more control over the custom ecosystem I have build for smart home privacy assessment as described below for example.

Network Monitoring ImposTer attempts to decrypt the TLS traffic from the devices and apps at various vantage points. However, the decryption fails in cases where apps or devices circumvent TLS interception using techniques such as certificate pinning [40, 52] as discussed earlier. During my experiments, out of approximately 20,000 requests send from my instrumented smart TV, 5.7% of them failed due to TLS handshake failures, for which a reason can be certificate pinning. Therefore, addressing TLS decryption failure to provide better coverage would be a plan for the future. Additionally, researchers have previously demonstrated security and privacy vulnerabilities in IoT protocols such as ZigBee [56] and BLE [43]. My future work would also include adding support for monitoring and intercepting these traffic as well. I also plan to improve the capabilities of my honeypot to capture and contextualize inter-device communication behaviours by extracting the semantics of the communications and building smart systems on top of that knowledge.
Device discovery My framework ImposTer showed evidence of various device discovery mechanisms employed by smart home devices. My future work would include understanding the privacy landscape of these discovery mechanisms with improved fuzzing techniques in a customizable honeypot in-order to capture and analyse requests.
Android proximity service Since Android 4.0, Android framework includes a high-level API called **Nearby** [14]. This library exposes APIs such as *NearbyConnection* and *NearbyMessage* to communicate with nearby Android or iOS devices irrespective of whether they are connected in the same network or not. Today, Google uses Nearby API in their exposure notification library [22]. I believe that, with minor configuration to my dynamic analysis scripts with Frida, I will be able to monitor Nearby APIs to characterize and understand PII violations while using this library.
Effects of countermeasures Analyzing the traffic generated by apps for the smart TV, I saw parameters such as *limit_ad_tracking* in the smart TV traffic. These parameters are meant to be selected by users to restrict tracking in their device. My future work would include validating the effectiveness of such countermeasures comparing with what apps have in their privacy policies especially when they are in a rich execution environment. This information can be utilized to access purpose of the communications and whether the data can be leaked for secondary advertising and tracking activities beyond those required to offer the service.
Automatization and fuzzing The app automation using Droidbot isn't perfect and this limits the coverage of my current framework. Many of the TV apps requires you to sign in with an account to access their full features. I plan to customize droidbot to bypass this limitation by signing up whenever required automatically using custom scripts and pre-created accounts. Additionally, I run each app for only 3 minutes. However, it would only be ideal to choose this timing based on a trail and error. My future work would include a testing phase for this as well. In addition, from the empirical results from the honeypot showcases the requirement for a smarter honeypot capable of crafting artificial interaction within the home network. This would require network level fuzzing to generate appropriate requests that IoT devices would respond to.

7 CONCLUSION

This thesis presents ImposTer, a unified privacy analysis framework for IoT platforms. By prototyping it and testing in real-world setups, I demonstrate the feasibility of characterizing smart home devices and their corresponding platforms. ImposTer stands apart from prior research due to its ability to monitor and characterize various smart home device and platform behaviours irrespective of the vendor. ImposTer empirically shows that prior black-box testing mechanism do not suffice to study the privacy and security risks of smart home ecosystem, automatically and exhaustively. ImposTer opens the ground for a new generation of black-box testing mechanisms for IoT devices with enhanced accuracy and coverage by enabling simulated but realistic IoT environments and device communications.

8 ACKNOWLEDGMENT

I'd like to sincerely thank my advisors for their continuous feedback cycle throughout this thesis and their patience to read through my early drafts. I'm extremely grateful for having Dr. Srdjan Matic in my team, for the countless number of hours spent together discussing experiments, methodology and providing me with the right guidance when needed. I would also like to thank Dr. Primal Wijesekera for providing me with ample amount of resources while instrumenting Android TV and making my life easier. Alvaro Feal and Julien Gamba, thanks for keeping my morale high whenever it deemed necessary. Finally, I thank our IT team at IMDEA Networks for providing the required equipment's for setting up the custom smart home within the lab.

REFERENCES

[1] [n. d.]. https://www.appcensus.io. ([n. d.]). "Accessed July 2021".
[2] [n. d.]. Amazon Fire TV. https://www.amazon.com/Amazon-Fire-TV-Family. ([n. d.]). "Accessed June 2021".
[3] [n. d.]. Androguard. https://androguard.readthedocs.io/en/latest/. ([n. d.]). "Accessed June 2021".
[4] [n. d.]. Android Developers, "App Manifest Overview,". https://developer.android.com/guide/topics/manifest/manifest-intro. ([n. d.]). "Accessed June 2021".
[5] [n. d.]. Apple Airtags. https://www.apple.com/es/airtag/. ([n. d.]). "Accessed June 2021".
[6] [n. d.]. Apple Find my. https://www.apple.com/icloud/find-my/. ([n. d.]). "Accessed June 2021".
[7] [n. d.]. Backsmali. https://github.com/JesusFreke/smali. ([n. d.]). "Accessed June 2021".
[8] [n. d.]. Best practices for unique identifiers. https://developer.android.com/training/articles/user-data-ids. ([n. d.]). "Accessed July 2021".
[9] [n. d.]. Developer Tools. https://developer.android.com/studio/. ([n. d.]). "Accessed June 2021".
[10] [n. d.]. Exodus Privacy. https://reports.exodus-privacy.eu.org/en/trackers/. ([n. d.]). "Accessed June 2021".
[11] [n. d.]. Get started with TV apps. https://developer.android.com/training/tv/start/start. ([n. d.]). Accessed July 2021.
[12] [n. d.]. Google Nest. https://www.home.nest.com. ([n. d.]). Accessed July 2021.
[13] [n. d.]. mitmproxy. https://mitmproxy.org. ([n. d.]). "Accessed June 2021".
[14] [n. d.]. Nearby Messages API. https://developers.google.com/nearby/messages/overview. ([n. d.]). Accessed July 2021.
[15] [n. d.]. Roku TV. https://www.roku.com/en-gb/products/roku-tv. ([n. d.]). "Accessed June 2021".
[16] [n. d.]. Usage of Android Advertising ID. https://support.google.com/googleplay/android-developer/answer/9857753. ([n. d.]). "Accessed July 2021".
[17] 2017. Alexa, are you listening? https://labs.f-secure.com/archive/alexa-are-you-listening/. (2017). Accessed July 2021.
[18] 2017. Google admits its new smart speaker was eavesdropping on users. https://money.cnn.com/2017/10/11/technology/google-home-mini-security-flaw. (2017). Accessed July 2021.
[19] 2017. 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). "Accessed July 2021".
[20] 2018. Amazon explains how Alexa recorded a private conversation and sent it to another user. https://www.theverge.com/2018/5/24/17391898/amazon-alexa-private-conversation-recording-explanation. (2018). Accessed July 2021.
[21] 2018. Android Things Platform. url-https://developer.android.com/things. (2018). "Accessed August 2021".
[22] 2020. Exposure Notifications API. https://developers.google.com/android/exposure-notifications/exposure-notifications-api. (2020). "Accessed July 2021".
[23] 2021. Android RPi. https://github.com/android-rpi. (2021). "Accessed June 2021".
[24] 2021. Everything You Need to Know About Apple's Find My Network Accessory Program. https://www.macrumors.com/guide/find-my-network-accessory-program/. (2021). Accessed July 2021.
[25] 2021. Frida.re. https://frida.re. (2021). "Accessed May 2021".
[26] 2021. HomeKit by Apple. https://www.apple.com/ios/home/. (2021). Accessed August 2021.
[27] 2021. How to Stop Your Smart TV From Spying on You. https://www.avast.com/c-smart-tv-spying-on-you. (2021). "Accessed June 2021".
[28] 2021. MCBrowserViewController. https://developer.apple.com/documentation/multipeerconnectivity/mcbrowserviewcontroller. (2021). Accessed July 2021.
[29] 2021. Smart Things by Samsung. https://www.smartthings.com. (2021). Accessed August 2021.
[30] 2021. Spotify Connect. https://support.spotify.com/us/article/spotify-connect/. (2021). Accessed July 2021.
[31] 2021. Use Nearby to find Chromecast devices. https://support.google.com/chromecast/answer/7073953?hl=en. (2021). Accessed July 2021.
[32] Razaghpanah Abbas, Nithyanand Rishab, Vallina-Rodriguez Narseo, Sundaresan Srikanth, Allman Mark, Kreibich Christian, and Gill Phillipa. 2018. Apps, Trackers, Privacy, and Regulators A Global Study of the Mobile Tracking Ecosystem. *Proceedings of the Network and Distributed System Security Symposium (NDSS).*
[33] Tawfiq Ammari, Jofish Kaye, Janice Y. Tsai, and Frank Bentley. 2019. Music, Search, and IoT: How People (Really) Use Voice Assistants. *ACM Trans. Comput.-Hum. Interact.* (2019).
[34] Leonardo Babun, Z. Berkay Celik, Patrick McDaniel, and A. Selcuk Uluagac. 2021. Real-time Analysis of Privacy-(un)aware IoT Applications. *Proceedings of the Privacy Enhancing Technologies Symposium (PETS).*
[35] Michael Backes, Sven Bugiel, Erik Derr, Patrick McDaniel, Damien Octeau, and Sebastian Weisgerber. 2016. On Demystifying the Android Application Framework: Re-Visiting Android Permission Specification Analysis. *Proceedings of the USENIX Security Symposium.*
[36] Alexander Benlian, Johannes Klumpe, and Oliver Hinz. 2019. Mitigating the Intrusive Effects of Smart Home Assistants by using Anthropomorphic Design Features: A Multi-Method Investigation. *Information Systems Journal* (2019).
[37] Hyunji Chung, Michaela Iorga, Jeffrey Voas, and Sangjin Lee. 2017. "Alexa, Can I Trust You?". *IEEE Computer* (2017).
[38] Daniel J. Dubois, Roman Kolcun, Anna Maria Mandalari, Muhammad Talha Paracha, David Choffnes, and Hamed Haddadi. 2020. When Speakers Are All Ears: Characterizing Misactivations of IoT Smart Speakers. *Proceedings of the Privacy Enhancing Technologies Symposium (PETS).*
[39] Jide S Edu, Xavier Ferrer-Aran, Jose M Such, and Guillermo Suarez-Tangi. 2021. SkillVet: Automated Traceability Analysis of Amazon Alexa Skills. (2021). arXiv:cs.CR/2103.02637

[40] Sascha Fahl, Marian Harbach, Thomas Muders, Lars Baumgärtner, Bernd Freisleben, and Matthew Smith. 2012. Why Eve and Mallory Love Android: An Analysis of Android SSL (in)Security. *Proceedings of the ACM Conference on Computer and Communication Security (CCS).*

[41] Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. 2016. Security analysis of emerging smart home application. *IEEE Symposium on Security and Privacy (SP).*

[42] Dennis Giese and Guevara Noubir. 2021. Amazon Echo Dot or the Reverberating Secrets of IoT Devices. In *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks.*

[43] Rohit Goyal, Nicola Dragoni, and Angelo Spognardi. 2016. Mind the Tracker You Wear: A Security Analysis of Wearable Health Trackers. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing.*

[44] Christian Kreibich and Jon Crowcroft. 2004. Honeycomb: Creating Intrusion Detection Signatures Using Honeypots. *SIGCOMM Comput. Commun. Rev.* (2004).

[45] Deepak Kumar, Riccardo Paccagnella, Paul Murley, Eric Hennenfent, Joshua Mason, Adam Bates, and Michael Bailey. 2018. Skill Squatting Attacks on Amazon Alexa. *Proceedings of the USENIX Security Symposium.*

[46] Yuanchun Li, Ziyue Yang, Yao Guo, and Xiangqun Chen. 2017. DroidBot: A Lightweight UI-Guided Test Input Generator for Android. In *Proceedings of the 39th International Conference on Software Engineering Companion (ICSE-C '17).*

[47] Kyeonghwan Lim, Jaemin Jeong, Seong-je Cho, Jongmoo Choi, Minkyu Park, Sangchul Han, and Seongtae Jhang. 2017. An Anti-Reverse Engineering Technique Using Native Code and Obfuscator-LLVM for Android Applications. In *Proceedings of the International Conference on Research in Adaptive and Convergent Systems.*

[48] Ziang Ma, Haoyu Wang, Yao Guo, and Xiangqun Chen. 2016. LibRadar: fast and accurate detection of third-party libraries in Android apps. In *Proceedings of the 38th international conference on software engineering companion.*

[49] Hooman Mohajeri Moghaddam, Gunes Acar, Ben Burgess, Arunesh Mathur, Danny Yuxing Huang, Nick Feamster, Edward W. Felten, Prateek Mittal, and Arvind Narayanan. 2019. Watching You Watch: The Tracking Ecosystem of Over-the-Top TV Streaming Devices. *Proceedings of the ACM Conference on Computer and Communication Security (CCS).*

[50] Alrawi Omar, Lever Chaz, Antonakakis Manos, and Monrose Fabian. 2019. SoK: Security Evaluation of Home-Based IoT Deployments. *IEEE Symposium on Security and Privacy (SP).*

[51] Andrea Possemato and Yanick Fratantonio. 2020. Towards HTTPS Everywhere on Android: We Are Not There Yet. In *Proceedings of the USENIX Security Symposium.*

[52] Abbas Razaghpanah, Arian Akhavan Niaki, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Johanna Amann, and Phillipa Gill. 2017. Studying TLS Usage in Android Apps. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT '17).*

[53] Joel Reardon, Álvaro Feal, Primal Wijesekera, Amit Elazari Bar On, Narseo Vallina-Rodriguez, and Serge Egelman. 2019. 50 ways to leak your data: An exploration of apps' circumvention of the android permissions system. *Proceedings of the USENIX Security Symposium.*

[54] N. Redini, A. Continella, D. Das, G. De Pasquale, N. Spahn, A. Machiry, A. Bianchi, C. Kruegel, and G. Vigna. 2021. DIANE: Identifying Fuzzing Triggers in Apps to Generate Under-constrained Inputs for IoT Devices. *IEEE Symposium on Security and Privacy (SP).*

[55] Jingjing Ren, Daniel J. Dubois, David Choffnes, Anna Maria Mandalari, Roman Kolcun, and Hamed Haddadi. 2019. Information Exposure From Consumer IoT Devices: A Multidimensional, Network-Informed Measurement Approach. *Proceedings of the Internet Measurement Conference (IMC).*

[56] Eyal Ronen, Adi Shamir, Achi-Or Weingarten, and Colin O'Flynn. 2017. IoT Goes Nuclear: Creating a ZigBee Chain Reaction. In *2017 IEEE Symposium on Security and Privacy (SP).*

[57] Said Jawad Saidi, Anna Maria Mandalari, Roman Kolcun, Hamed Haddadi, Daniel J. Dubois, David Choffnes, Georgios Smaragdakis, and Anja Feldmann. 2020. A Haystack Full of Needles: Scalable Detection of IoT Devices in the Wild. In *Proceedings of the ACM Internet Measurement Conference.*

[58] Janus Varmarken, Hieu Le, Anastasia Shuba, Athina Markopoulou, and Zubair Shafiq. 2020. The TV is Smart and Full of Trackers: Measuring Smart TV Advertising and Tracking. *Proceedings of the Privacy Enhancing Technologies Symposium (PETS).*

[59] Xueqiang Wang, Yuqiong Sun, Susanta Nanda, and XiaoFeng Wang. 2019. Looking from the Mirror: Evaluating IoT Device Security through Mobile Companion Apps. *Proceedings of the USENIX Security Symposium.*

[60] Primal Wijesekera, Arjun Baokar, Ashkan Hosseini, Serge Egelman, David Wagner, and Konstantin Beznosov. 2015. Android Permissions Remystified: A Field Study on Contextual Integrity. *Proceedings of the USENIX Security Symposium.*

[61] Primal Wijesekera, Arjun Baokar, Lynn Tsai, Joel Reardon, Serge Egelman, David Wagner, and Konstantin Beznosov. 2018. Dynamically Regulating Mobile Application Permissions. *IEEE Symposium on Security and Privacy (SP)* (2018).