TU WIEN Informatics

Investigating Data Privacy and Network Communication in IoT Devices and Their Apps across Android and iOS

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Martina Komsic, BSc
Matrikelnummer 01634065

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Associate Prof. Dipl.-Ing. Dr.techn. Martina Lindorfer, BSc
Mitwirkung: Carlotta Tagliaro, MSc

Wien, 3. September 2025

Martina Komsic

Martina Lindorfer

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

TU WIEN Informatics

Investigating Data Privacy and Network Communication in IoT Devices and Their Apps across Android and iOS

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Software Engineering & Internet Computing

by

Martina Komsic, BSc
Registration Number 01634065

to the Faculty of Informatics

at the TU Wien

Advisor: Associate Prof. Dipl.-Ing. Dr.techn. Martina Lindorfer, BSc
Assistance: Carlotta Tagliaro, MSc

Vienna, September 3, 2025

Martina Komsic	Martina Lindorfer

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Martina Komsic, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, haben ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT- Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 3. September 2025

Martina Komsic

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to Carlotta Tagliaro for her invaluable guidance and support throughout the course of this thesis. Her expertise and insightful feedback were instrumental in shaping both the direction and the outcome of this work.

I am also sincerely thankful to my thesis advisor, Martina Lindorfer, for her guidance and for providing the infrastructure and IoT devices that formed the basis of this thesis.

Finally, I want to thank my family and friends for their support during this journey.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

Consumer Internet of Things (IoT) devices offer unprecedented convenience by automating and improving everyday tasks, from managing home security systems to optimizing energy usage, making them essential to modern households. Despite these devices' convenience, they collect and transmit significant amounts of personal data, often without user awareness.

We provide the first extensive cross-platform analysis of IoT devices and their companion mobile apps on Android and iOS across different network settings, focusing on the privacy implications for users. We design and implement CapIoT, a semi-automated framework, that enables systematic capture and replay of interactions between IoT devices and their companion apps. Using CapIoT, we test ten IoT devices, collecting and analyzing traffic at both the device and app level. Using the collected traffic, we aim to understand how devices and apps communicate in different network scenarios across mobile platforms, what kinds of data are shared, and with whom.

Specifically, our analysis covers device and app traffic, endpoint diversity, protocol adoption, reliance on cloud providers, and handling personally identifiable information (PII). We also want to understand the extent of data sharing with third-party trackers and analytics services. Our findings reveal that while none of the devices directly contact these services, most companion apps do, highlighting their significant role in privacy risks. We also examine how network settings and mobile platforms impact communication patterns, noting substantial differences.

We underscore the need for greater transparency, robust data protection measures, and strict regulatory enforcement in the IoT ecosystem.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 1

Introduction

1.1 Motivation

IoT devices are vital in our current society. They have entered our houses and are helping us with everyday tasks, ranging from vacuuming the floors to smart alarm systems. Consumer spending on smart home products and services is projected at 170 billion dollars by 2025, highlighting their pervasiveness in our society [54]. To this end, they collect, process, and store a conspicuous amount of data about the surrounding environment and the users themselves. Such information must be handled securely and with users' privacy in mind. However, data leaks are far from being unprecedented. In 2023, Anker, a popular consumer devices vendor, admitted that their Eufy smart camera streams were not always encrypted [25]. An attacker could just access the recordings and potentially plan a robbery, knowing when users were not home.

IoT devices are not the only players in this ecosystem. Companion apps can help mediate the communication between the devices and (cloud) endpoints and become helpful in operating the devices without being in their proximity. Nearly every device has a companion app, available for Android on the Google Play Store and iOS on the Apple App Store. Consequently, companion apps process and store sensitive information collected from the devices and information about the cloud endpoints. If the data is not correctly handled in the companion apps, it could threaten the correct functioning of the devices themselves. The SmartTub iOS and Android app, used to control Jacuzzis and other hot tubs, leaked customer details and allowed malicious actors to play with the tub setting, changing the hydrojets' intensity and water temperature, potentially hurting the consumer [70].

Previous work focused on analyzing companion apps with a combination of static analysis and fuzzing and found that they embed a range of sensitive information, e.g., API keys or passwords [52, 63, 69]. Schmidt et al. showed how they could reconstruct the key

used for 3DES encryption of account credentials from a companion app associated with a smart camera, thus defeating confidentiality [52]. Researchers have also analyzed the traffic generated by IoT devices to recognize event-based patterns and look for tracking and advertisement [31, 39, 45, 58].

However, the literature lacks an understanding of the correlation between companion apps and device communication. We bridge studies on companion apps and device traffic, comparing and better understanding the role of mobile apps and how they enrich communication. Companion apps not only help control the devices but can also enrich the information they get from them. As there is no "protection" for the data the apps receive, they can freely combine other data and fingerprints they already have from the user. If shared with third-party services like trackers and analytics endpoints, this can severely impact users' privacy. As devices can directly connect to the Internet without relying on the app to forward data, we are interested in the information flow. In this thesis, we investigate the interplay between companion apps and IoT device communication. We use our IoT testbed and two smartphones to launch over 200 automated experiments on 10 IoT devices.

1.2 Research Questions

We analyze whether there is a difference in the endpoints that devices and apps contact, what information they exchange, and how it differs between the two players. Previous research showed the wide usage of mobile app advertisements (ads) and trackers for user targeting [46, 50, 59]. Related studies showed that even the devices include ads [31, 49, 60], part of which might be specific to the IoT ecosystem. We try to understand whether apps contact more trackers than devices, what information about the users is collected, and if performing a certain action on the device (e.g., switching a smart alarm system on/off) can trigger requests to such services.

Secondly, we investigate how communication patterns change when network settings change. Guidelines from ENISA [15] or NIST [36] recommend choosing local communication when possible without letting traffic flow through remote (cloud) endpoints. We assess this by comparing the traffic when companion apps and devices are connected to different networks (WAN) versus the same one (LAN).

Thirdly, we investigate the information shared by companion apps by capturing phone traffic to identify personal identifiable information (PII), including (1) *Device Identifiers*, e.g., IMEIs, Advertiser ID, MAC addresses, SSIDs, (2) *User Identifiers*, e.g., name, email address, phone numbers, (3) *Location*, e.g., geographical coordinates and IPs, and (4) *Credentials*, i.e., username and password. Further, we look at any IDs (e.g., Android IDs, Google Advertising IDs) that can identify a specific user and determine whether they are reused across devices and endpoints. If this is the case, depending on the information they carry and the contacted parties, they could pose serious risks to users' privacy by making them the target of more precise profiling.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Lastly, we analyze platform-specific differences between Android and iOS communication patterns and their impact on user privacy.

Specifically, we are going to answer the following research questions:

- **[RQ1] Which network endpoints–identified by domain, port, country, provider, and category–are accessed by IoT devices and their companion apps?**
- **[RQ2] Does the communication pattern change in the different network scenarios?**
- **[RQ3] What specific data is collected and shared?**
- **[RQ4] What endpoint and privacy-related differences exist between the Android and iOS versions of the companion apps?**

In summary, we make the following contributions:

- We design and develop CapIoT, a semi-automated framework that records and replays interactions with IoT companion apps on both Android and iOS. We test 10 devices and their companion apps under different network settings (LAN vs. WAN).
- We show that while no devices contact *Tracking & Analytics* endpoints, 7 companion apps (70%) do so. We identify apps as the main culprit in introducing tracking in the IoT ecosystem.
- We evaluate LAN versus WAN communication, revealing that 30% of the device manufacturers do not follow ENISA guidelines of preferring local communication when possible. Instead, they route all traffic through remote endpoints, even when local channels are available.
- We analyze the transmitted data for any PII. We identify one case where the same advertiser ID is shared across two devices, thus providing aggregated data for user tracking. We also find that LAN scenarios often trigger more PII sharing than WAN.
- We provide a cross-platform comparison of Android and iOS in the IoT ecosystem. We find that Android apps communicate with various endpoints, often including Google services and third-party trackers. In contrast, iOS apps use fewer endpoints overall but depend more strongly on vendor cloud servers.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 2

Related Work

IoT Companion Apps. Recent work investigated IoT companion apps [6, 8, 47, 49] looking for security and privacy issues. Zhou et al. [68] studied the interactions between IoT devices, cloud, and apps using state machines and found invalid transitions, leading to device hijacking. Nevertheless, they only relied on companion apps' code. Wang et al. [63] analyzed companion apps without the need for the device, focusing on rebranding and propagation of old vulnerabilities. No studies focus on the distinction of device- and app-generated traffic, whether the contacted endpoints are the same, and how the interaction changes in case the mobile phone (on which the app is installed) and the device are connected to different networks. Thus, we aim to spot such differences.

IoT Backends. Tagliaro et al. analyzed the security of IoT backends, specifically those using IoT-focused protocols such as MQTT, CoAP, and XMPP, and found critical vulnerabilities, including insecure transport protocols, weak authentication, and susceptibility to DoS attacks. Notably, 99.84% of MQTT and XMPP backends lacked secure transport protocols, with only 0.16% adopting TLS, and over 70% of those used vulnerable versions [57].

IoT Device Privacy. Several studies focused on identifying IoT device activities (e.g., turning on/off) by looking at traffic characteristics at the network level as well as packet signatures [10, 33, 39, 42, 55, 58]. Loi et al. [30] investigated 20 common consumer devices along four dimensions: confidentiality, integrity, access control, and reflective attacks. They found five devices communicating over plaintext with their app and sharing sensitive information. When consumers are asked how they feel about the privacy of their devices, contrasting opinions emerge. While Emami-Naeini et al. [14] show that privacy ranks as a top factor in purchase decisions, Zheng et al. [67] claim users prefer functionality over privacy and are willing to share their data with third parties if they can benefit from it. Studies also analyzed IoT device traffic, identifying event patterns, tracking, and advertisements [31, 39, 45, 58]. Further, mobile apps and IoT devices were

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

found to widely use ads and trackers for user targeting [46, 50, 59], with some ads being unique to the IoT ecosystem [31, 49, 60].

Android vs iOS. Kollnig et al. analysed 24,000 popular free apps (12,000 Android, 12,000 iOS) using static code inspection and a Machine-In-The-Middle capture of each app's first-launch traffic. They observed at least one tracking library in 89% of Android and 79% of iOS apps. Google receives data from almost every Android app and nearly two-thirds of iOS ones, while 55% versus 31% leak the advertising identifier. iOS apps, however, request sensitive permissions such as Location (49% iOS vs. 28% Android) more often. Thus, neither ecosystem offers clearly better privacy: Android leaks identifiers more aggressively, whereas iOS demands wider access [28]. However, the study focuses on trackers, permissions, and advertising identifiers at first launch. It does not compare where apps and devices connect or how they communicate. We close this gap. We analyze endpoints (domains, providers, and countries), ports and protocols, and overall communication behavior. We compare Android and iOS companion apps for the same ecosystems. We also relate app traffic to device traffic and concrete user actions and test how different network placements change these patterns.

Much of the prior work studies individual parts of the IoT ecosystem (IoT backends, mobile apps, or devices) in isolation. However, the literature lacks an understanding of the correlation between companion apps and device communication. We bridge this gap by jointly analyzing app and device traffic, better understanding the role of apps and how they enrich communication.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 3

Methodology

We answer our research questions by proposing a tool, CapIoT (for a complete description see Chapter 6), to automatically test the communication of IoT devices and their companion apps. We need a replicable testing procedure to avoid tedious manual effort. We adopt a system of screenshots and coordinates on the phones' screen to record our manual interaction with the companion apps, which we then make easily repeatable by simulating touch events on the screen. In this way, we only need to test the apps once manually, but we can re-run the recorded interaction multiple times, similar to Mandalari et al.'s approach [31]. We record the traffic generated by the phone and IoT device with *tcpdump* [22].

Further, as we notice that most of the collected traffic is indeed encrypted, we employ *mitmproxy* [34] in transparent mode and *frida* [19] to bypass certificate pinning and Machine-In-The-Middle (MITM) traffic sent to and from the companion apps. We can only capture analyzable app-generated traffic, as we would be required to reverse each device's firmware and install custom certificates to do the same for device traffic, with the high risk of bricking the devices themselves. We deem this a reasonable limitation. We show our methodology in Figure 3.1.

We parallelize both experiments on two phones and perform all our analyses between July and August 2025.

Providers Analysis. We first gather the IP ranges for the ten major (IoT) cloud providers [66] and determine for each endpoint if its associated IP addresses belong to one of the providers.

Reverse DNS Lookup. Before moving on to the analysis, we check for unmatched IP addresses, i.e., IPs for which we could not find the resolving hostname in the DNS responses we captured. We run reverse DNS queries from the Domaintools [11] database to gain further insights into the unmatched IPs. In case of a hit, for each IP, we look

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

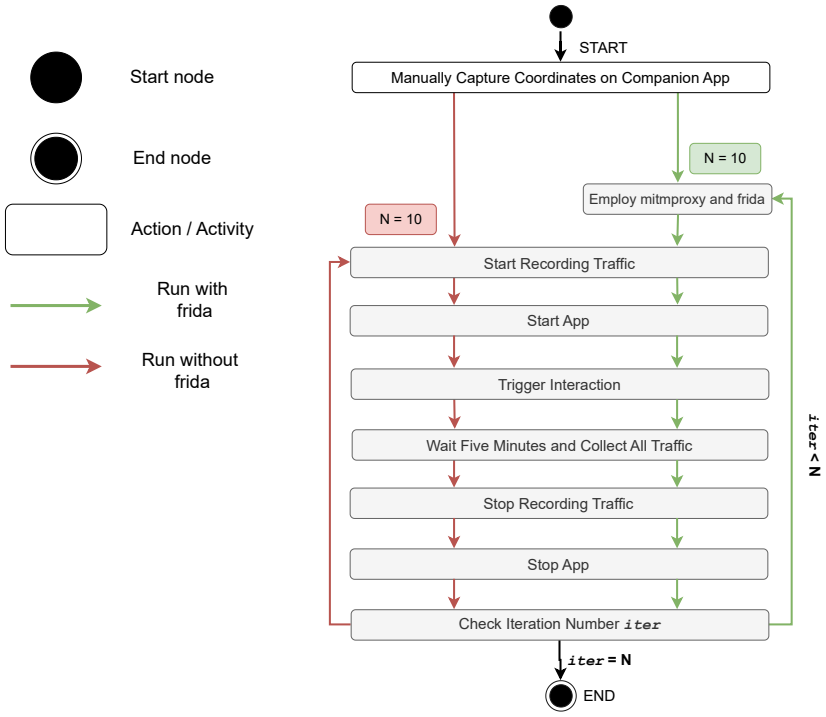


Figure 3.1: High-level overview of our experimental procedure. Each experiment consists of ten iterations without instrumentation and ten iterations with *mitmproxy* and *frida*.

for full domain matches between the hostnames returned by the queried services and the endpoints we previously obtained through DNS responses at device granularity, i.e., we only check if the returned endpoint matched one of the hostnames we found for the specific device(s) contacting the IP address. If we cannot find a full match, we match the Top Level Domain (TLD) with the same approach. Two researchers independently verified the results and resolved conflicts. In some cases, we could not find a reasonable match; thus, we leave the IP as "unresolved." We acknowledge that this approach might present some accuracy limitations, but we deem it essential to gain more insight into the communication patterns of IoT devices and their respective apps. Further, it allows us to run a classification on the endpoints.

Endpoints Categorization. We investigate who companion apps and IoT devices communicate with by categorizing the contacted endpoints to spot potentially problematic ones such as trackers. As we cannot classify IP addresses, we remove them from the analysis. We want to understand whether one of the two players contacts more tracking and analytics domains than the other and, generally, how the categories distribution looks across them. To this end, we adapt the approaches by Ren et al. [49, 50], which they compiled from ad-blocking lists and by flagging endpoints as first/support/third party, and by employing the Exodus tracker list [16]. Our categorization consists of three

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

labels: (1) *First Party* if the endpoints directly belong to the specific IoT device vendor, (2) *Support Party* if the endpoints are needed to offer extra functionalities, e.g., login and account management, or belong to known cloud hosting providers, (3) endpoints that are associated with *Tracking & Analytics*, that might collect and analyze users' data to provide personalized advertisement or content. Given the dynamism of the tracking and advertisement ecosystem and that the categories from Ren et al. might be outdated, we only use them as a baseline for our classification. Two independent researchers manually check for mislabeled categories by looking at what services the organizations owning the endpoints offer or whether the endpoints are included in well-known tracking blocklists [5, 24]. For example, if an organization offers marketing insights into customer activities, we flag its endpoint as "Analytics" (e.g., Google Analytics). In case of conflicts in the labeling, a third researcher intervened.

Certificate Pinning in Apps. To understand how many companion apps employ certificate pinning to verify the certificates' authenticity, we run the static-analysis stage of HALY by Steinböck et al. [56]. Concretely, HALY looks for the inclusion of certificates or the hashes of the certificates.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 4

Background

In this chapter, we provide the necessary background information to understand the design of our experiments and the interpretation of the results. We first introduce IoT communication patterns and control modes, then discuss common application-layer protocols, and finally describe the tools and techniques used to decrypt app traffic.

4.1 IoT Communication Patterns

In general, there are different ways in which IoT devices can communicate with other devices or entities (e.g., cloud backends). Figure 4.1 shows four different communication types [32]:

1. **Device-to-Device (D2D)**: IoT devices communicate directly with each other without involving an application or cloud backend.
2. **Device-to-Application (D2A)**: Communication occurs between an IoT application (e.g., installed on a phone) and IoT devices.
3. **Device-to-Gateway (D2G)**: A gateway acts as a hub in this type of communication. The gateway is placed near the IoT devices and enables communication between IoT devices and, for example, a cloud backend.
4. **Device-to-Cloud (D2C)**: IoT devices and the cloud backends communicate directly with one another (e.g., sending status updates or receiving commands). The cloud backend also exchanges messages with IoT applications (e.g., receiving device status updates or sending commands).

The communication type heavily depends on the IoT device. A device without Wi-Fi or Ethernet capabilities cannot use D2C communication. Instead, it relies on a gateway to

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

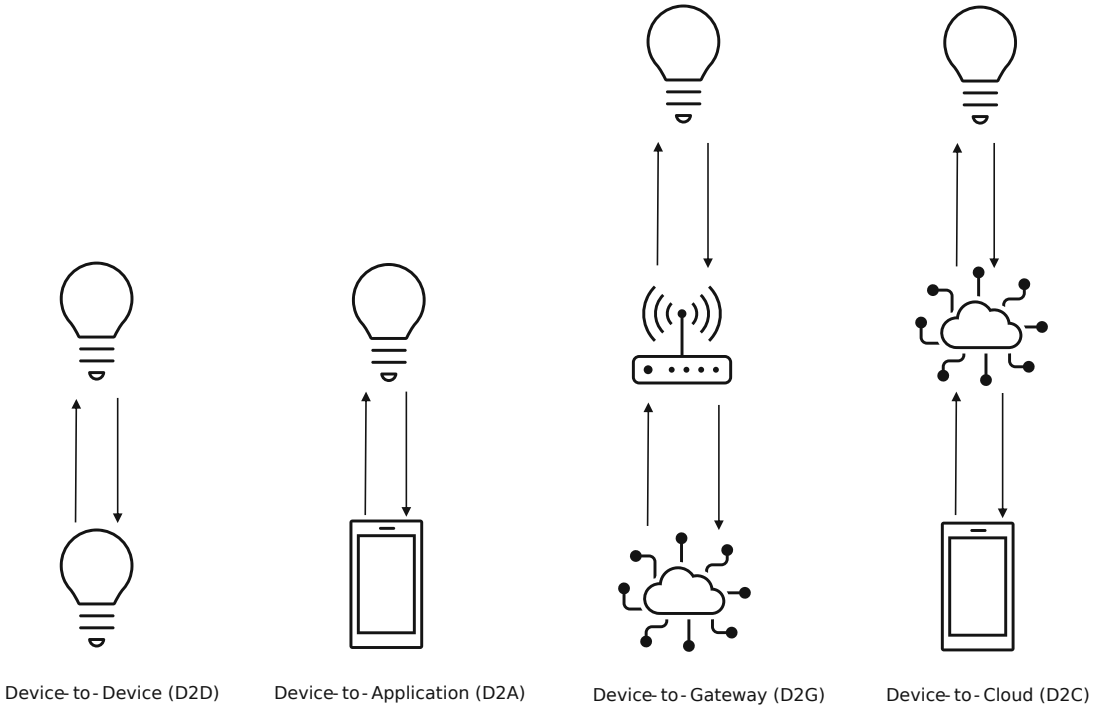


Figure 4.1: IoT communication types

forward the messages to the cloud. In this thesis, we analyse Device-To-Application and Device-To-Cloud communication.

In addition to looking at how devices communicate, we also consider how users control the devices. We distinguish four control modes [27]:

1. **Local Area Network (LAN)**: Companion app and IoT device are on the same local network and can communicate directly (e.g., over Wi-Fi, Bluetooth, or Zigbee). Some IoT devices operate only in this mode. For example, a Chromecast typically expects the phone to be on the same LAN. In this setting, the control mode typically maps to **D2A**. Some vendors nonetheless relay traffic via the cloud (**D2C**) even on LAN. However, security guidelines (e.g., ENISA [15]) recommend preferring local communication over remote if possible.

2. **Wide Area Network (WAN)**: Companion app and IoT device are on different networks (e.g., when the user is away from home and wants to control the device remotely). Local communication is not possible, so they communicate through the cloud. This control mode corresponds to **D2C**.

3. **Physical**: The user interacts with the device through built-in buttons or switches.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

4. **Multimodal**: The device supports multiple input channels (e.g., app, voice assistant, and sensor-based triggers).

In this thesis, we focus on LAN and WAN control. For this reason, all selected devices support Wi-Fi connectivity and can operate both in LAN and WAN mode, so that we can trigger actions (e.g., on/off, volume changes) automatically via a script and compare the resulting traffic across the two scenarios. Devices that only offer physical control are out of scope because they are difficult to automate reliably. Some devices in our dataset are multimodal (e.g., *EchoDot v5*), but we interact with them exclusively through their companion apps.

4.2 IoT Application-Layer Protocols

IoT devices are generally limited by memory size, processor capabilities, and energy capacity. Therefore, a lightweight communication protocol that does not use too many resources is needed. It also should be able to work in unreliable and low-bandwidth networks [2]. Survey data show that MQTT best fits these constraints and has become the dominant IoT messaging choice. According to the 2024 Eclipse IoT & Embedded Developer Survey, 56 % of respondents use MQTT in production, well ahead of HTTP/HTTPS (32 %), raw TCP/IP sockets (30 %), and MQTT + Sparkplug (8 %). All other IoT-specific protocols like CoAP, AMQP, and XMPP are not mentioned in the survey [13]. Guided by these observations and our findings during the experiments, we describe MQTT and XMPP in the remainder of this section.

4.2.1 MQTT

Core Idea. Message Queueing Telemetry Transport (MQTT) follows a publish-subscribe model. Clients do not talk to each other directly; instead, they publish messages to topics on a broker. Any client that subscribes to a topic receives the messages for that topic. Clients can subscribe to and publish to multiple topics. A broker routes the messages. The cloud, IoT device, and mobile apps are the main components of a cloud-based IoT system. In the case of IoT cloud communication, the cloud takes the broker's role. The devices and mobile applications act as the clients publishing and subscribing to topics [26]. The communication between clients proceeds as follows: the publisher sends a message including the topic to the broker. The broker then sends the message to all clients who have subscribed to that topic [12]. The topic can also have hierarchically structured paths like *Bedroom/lamp*. The forward-slash (i.e., "/") separates topic levels. The MQTT protocol also supports two wildcards: "#" and "+". The number sign (i.e., "#") is a multi-level wildcard, and the plus sign (i.e., "+") is a single-level wildcard. A client can subscribe/publish to multiple topics by using structured paths and wildcards. Suppose a user has numerous lamps in the bedroom and wants to turn them off. Instead of sending publish messages to every topic (e.g., *Bedroom/lamp*, *Bedroom/lamp1*, etc.), the client

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

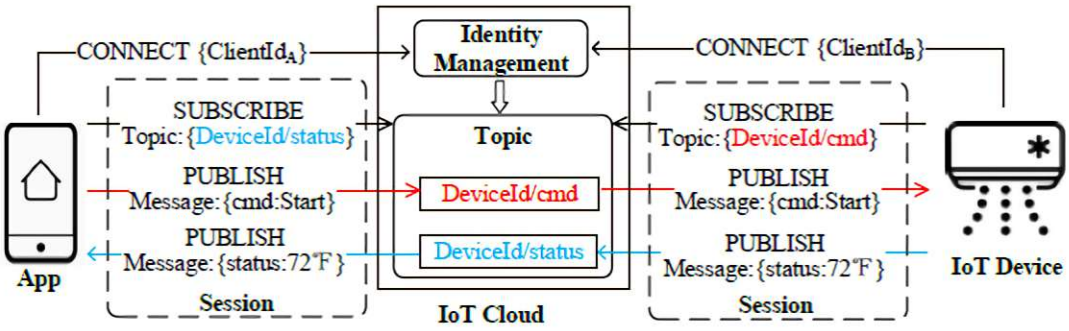


Figure 4.2: MQTT communication between clients. Reprinted from Jia et al. [26]

(e.g., app) would send only one publish message to the "Bedroom/+" topic. The broker will forward the message to all topics matching the wildcard [38].

Example. Figure 4.2 illustrates a typical flow. To establish a session, clients (e.g., mobile applications or smart bulbs) must first send a CONNECT message to the broker, including their ClientID. After establishing the session, the clients can send PUBLISH and SUBSCRIBE messages. For example, a mobile app would subscribe to */DeviceID/status* to receive state information. A smart air conditioner would send a SUBSCRIBE message including the topic */DeviceID/cmd* to receive commands. To turn on the air conditioner, the app would send a PUBLISH message including the command (e.g., "start") to the topic */DeviceID/cmd*. The air conditioner, which subscribed beforehand to the topic, would receive this message and act accordingly. In return, the air conditioner periodically publishes its state to the topic */DeviceID/status* so that the subscribers (e.g., the app) are up-to-date.

Features. MQTT offers delivery guarantees via Quality-of-Service (QoS) levels (0–no ack, 1–single ack, 2–two way ack), retained messages so late subscribers get the last known state, and a "Last Will" to notify others if a client drops unexpectedly. Deployments typically secure MQTT with TLS and authenticate clients using passwords or certificates [38].

4.2.2 XMPP

Core Idea. The eXtensible Messaging and Presence Protocol (XMPP) is based on the client-server model and runs over TCP with TLS and Simple Authentication and Security Layers (SASL) for security. The architecture and message exchange are similar to the email network. Clients address each other using Jabber IDs (JIDs) like *user@xmpp.server.com*, and servers relay messages using server-to-server federation, similar to email. Messages are exchanged using XML stanzas, which are sent over an XML stream [51].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

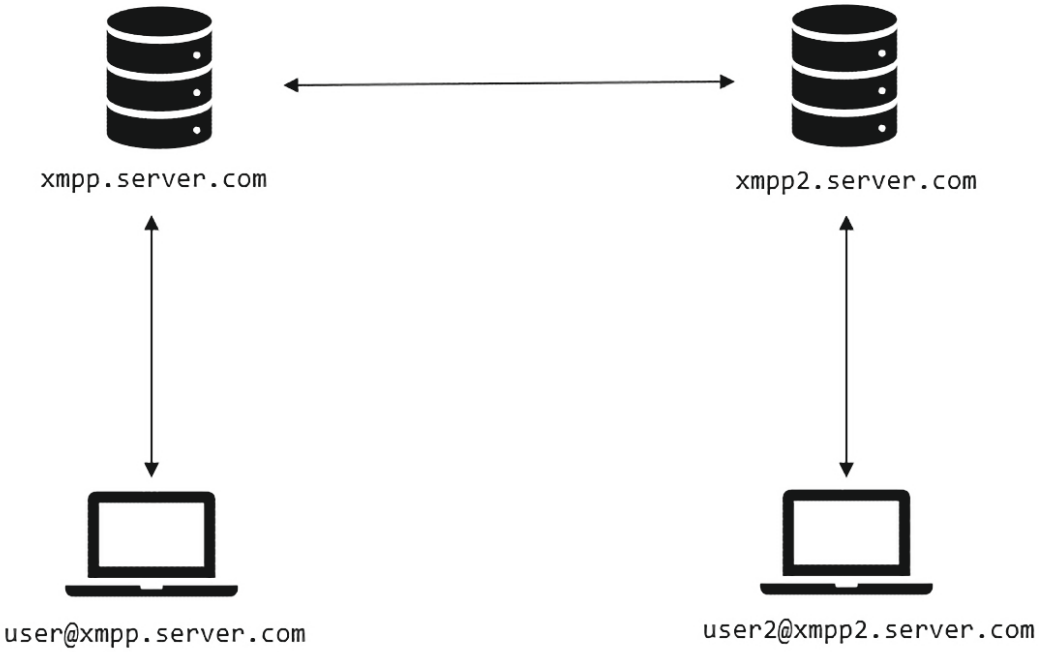


Figure 4.3: XMPP Client-Server architecture. Adapted from Saint-Andre [51]

Example. Figure 4.3 illustrates a typical message exchange. The user with the address *user@xmpp.server.com* wants to send a message to user *user2@xmpp2.server.com* associated with a different server. The message is first sent to the server *xmpp.server.com* since that entity manages the sender. The server then forwards the message to the server associated with the recipient. In this case, the message will be sent to the server with the address *xmpp2.server.com*. At last, the server forwards the message to the right recipient. Since communication with another client always happens through a server, XMPP does not support direct Device-to-Device (D2D) communication. It instead uses a client-to-server-to-server-to-client approach [51].

Features. Besides the core functionality, XMPP provides XMPP Extension Protocols (XEP). These protocols extend the core specification with additional functionality. For example, extensions support multi-user chat or the publish/subscribe architecture. Especially, the publish/subscribe extension is helpful when using XMPP for IoT applications. In addition, XMPP has an inbuilt access control mechanism [62].

4.3 Decrypting App Traffic with mitmproxy and frida

We need to decrypt the HTTPS traffic to analyse which PII the apps share and collect. We use *mitmproxy* for intercepting and decrypting HTTPS flows and rely on *frida* to

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

bypass certificate pinning that would otherwise block the proxy.

mitmproxy. *mitmproxy* is an open-source TLS proxy that sits between the app and the server. To the app, we look like the server, and to the server, we look like the client, allowing us to read both sides of the conversation. However, the Certificate Authority (CA) system is designed to stop exactly this attack. During the TLS handshake, the client accepts the server certificate only if it is signed by a CA that the operating system root store already trusts. If the signature is missing or is produced by an unknown CA, the client drops the connection. To overcome this, *mitmproxy* provides a full CA implementation that forges server certificates on the fly for every hostname and signs each certificate with its root key. We manually add the *mitmproxy* root certificate to the phone's trust store so the client treats every forged certificate as genuine. With the root CA in place, *mitmproxy* can decrypt and modify all HTTPS traffic and export TLS session keys for offline analysis in Wireshark. We run *mitmproxy* in *transparent mode*. Outbound packets from the phone are redirected to the proxy using *iptables* rules, so the app needs no proxy settings. Figure 4.4 illustrates the request flow [35]:

1. The client opens a TCP connection to the server.
2. The *iptables* rules redirect that connection to *mitmproxy*.
3. The client initiates the TLS handshake and, as part of that handshake, sends a Server Name Indication (SNI) that contains the hostname it wants to reach.
4. *mitmproxy* contacts the real server and starts its own TLS handshake, forwarding the same SNI it just received from the client.
5. The server returns its certificate chain. *mitmproxy* extracts the Common Name (CN) and Subject Alternative Names (SANs) from the certificate.
6. *mitmproxy* forges a certificate with the same CN and SANs, signs it with its built-in CA, and resumes the paused handshake with the client.
7. The client now sends its HTTPS request to *mitmproxy*.
8. *mitmproxy* forwards the request over its separate TLS session to the real server.

Certificate Pinning. During a TLS handshake, the server sends a certificate chain, which is an ordered list where each certificate is signed by the one above it. A client accepts the chain if every signature is valid and the root certificate appears in the operating system root store. Because Android and iOS ship with built-in root certificate stores, most apps trust the system's own list. Certificate pinning takes a different approach. Instead of trusting every root CA in the OS store, the developer embeds "pinned" certificates or the corresponding public key hash directly in the app package. When the app connects to its server, it accepts the TLS handshake only if the pinned certificate (or key hash) is in the chain. Certificate pinning blocks attacks that abuse a compromised or misconfigured root

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

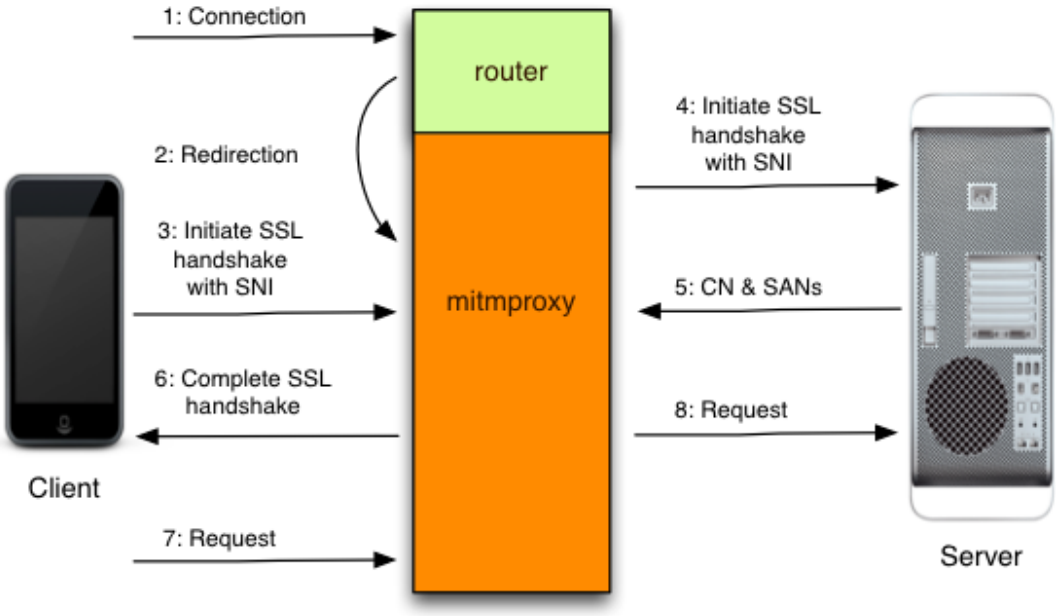


Figure 4.4: Transparent HTTPS interception with *mitmproxy*. Reprinted from *mitmproxy* [35]

certificate and prevents MITM proxies from substituting their own certificates during the TLS handshake [43]. Steinböck et al. analyzed 2,646 popular apps (Android and iOS) and found that 25.6% of Android apps and 28.6% of iOS apps contained embedded certificates or hashes [56]. Using HALY's methodology, we test our companion apps and find evidence of certificate pinning in all but three apps: *com.meross.meross* (Android & iOS), *com.wizconnected.wiz2* (Android & iOS) and *com.etekcity.vesyncPlatform* (iOS). While pinning provides an extra layer of protection, it complicates traffic analysis. Tools such as *mitmproxy* generate their own server certificates, which the certificate pinning checks in the app rejects. In our experiments, we bypass this check at run time with *frida* to still inspect the encrypted traffic.

frida. *frida* [19] is a dynamic instrumentation toolkit that lets us hook into methods of a running app on Android or iOS. Rather than writing our own hooks, we rely on SSL-unpinning scripts [1, 53] that automatically replace certificate verification functions. When we attach *frida* to the companion app, the scripts patch methods such as *checkServerTrusted* on Android so that every check returns trusted. With pinning disabled, the app accepts the certificate that *mitmproxy* presents, and we can inspect the previously encrypted traffic without modifying the app binary.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 5

IoT Setup

We build the IoT testbed shown in Figure 5.1 to have a controlled environment. It consists of the following components: (1) a local Access Point (AP) that provides IP connectivity from the ISP to the IoT devices and phones and captures all the network traffic; (2) a remote AP, which provides IP connectivity from the ISP only to the phone for WAN experiments, (3) the IoT devices under test, all connected to the Internet via the local AP; (4) Two phones, one Android and one iOS, connected to the Internet through the local or remote AP, with the companion apps of the IoT devices installed, and controllable via scripts; (5) CapIoT, running at the local or remote AP, which interacts with the companion apps. For more details on CapIoT refer to Chapter 6. We present more details on each testbed component below.

Phones. We employ a rooted Google Pixel 6A phone with Android 16 and an Apple iPhone 8 running iOS 16.11.7 that we jailbreak using palera1n v2.0.2 [41]. We equip both phones with *frida-server* v17.2.12 (Android) and v16.11.7 (iOS) binaries [19].

IoT Devices. We provide a complete list of the 10 devices we test in Table 5.1 together with the functionality we trigger. All devices support Wi-Fi connections and have a corresponding companion app (which we install on both phones).

Local AP. Our test environment uses an Intel NUC 13 Pro running Ubuntu 24.04.2. The AP provides Internet connectivity for the IoT devices using two different Wi-Fi card adapters. One for Android experiments and the other for iOS experiments. We configure the AP with our custom tool, CapIoT, which captures network traffic from IoT devices and enables automated experiments by simulating user interaction through touch events.

Remote AP. We use another machine running Ubuntu 24.04.2, connected to a different network than our primary AP. We install the same software as on the local AP. It provides Internet to the phones using the built-in Wi-Fi module to test the WAN scenario.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

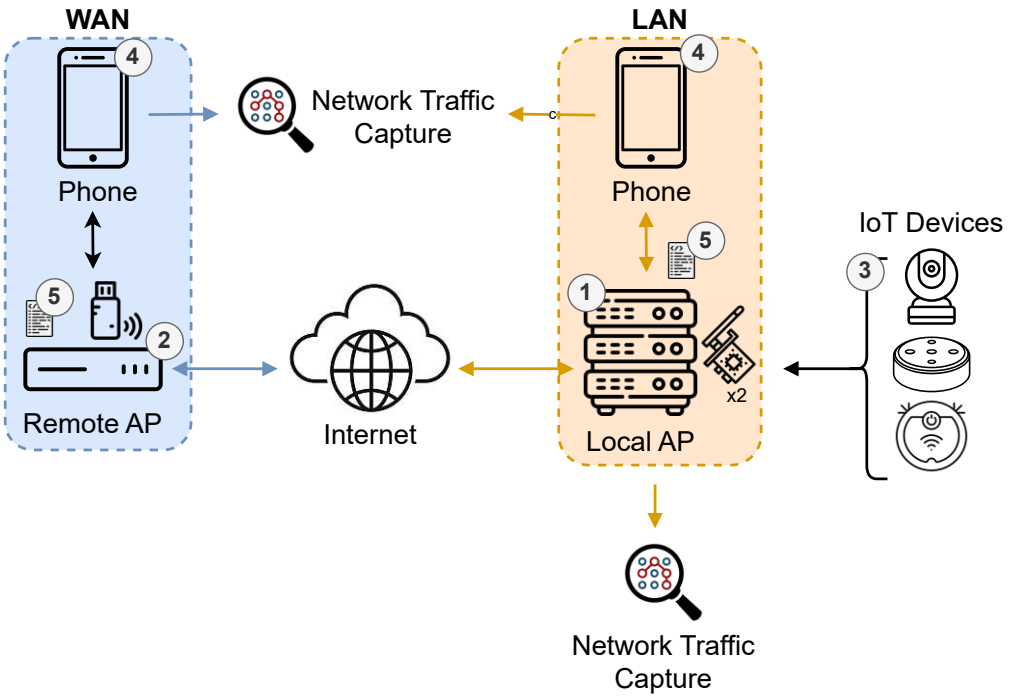


Figure 5.1: Experimental setup for WAN and LAN scenarios. In the LAN case, the phone (4) connects to IoT devices (3) through the local AP (1), while in the WAN case, the phone (4) communicates via a remote AP (2) over the Internet. Network traffic is captured in both settings, and logs (5) are collected for analysis.

5.1 Server Preparation

We prepare the server by installing essential tools and configuring the DHCP service to ensure static IP addressing. Table 5.2 summarizes the software installed on the server. Python dependencies required for our tool CapIoT are specified in the *pyproject.toml* and therefore not mentioned in the table.

5.1.1 DHCP Configuration

To ensure consistent IP addressing, we configure the DHCP server to assign static IP addresses based on the MAC addresses. We first verify that the phones use their actual hardware MAC addresses by turning off MAC address randomization in the Wi-Fi settings. We then modify the *dnsmasq* configuration. We add a line for each phone specifying its MAC address, hostname, and desired IP address. We show an example configuration entry in Listing 5.1.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Table 5.1: Overview of the IoT devices we test, grouped by category. For each device we list the corresponding companion app on Android and iOS, including the app identifier and version used during the experiments.

	Device	Android App ID (Version)	iOS App ID (Version)
Functionalities: microphone on/off			
Cam	Teckin TC100	com.lp.teckin (1.2.5)	1492022146 (1.2.8)
	Philips SCD923	com.philips.ph.babymonitorplus (1.4.0)	1544760744 (1.4.2)
Functionalities: turn on/off			
Light	Xiaomi MJDPL01YL	com.xiaomi.smarthome (10.6.626)	957323480 (10.6.202)
	WiZ A67	com.wizconnected.wiz2 (1.24.3)	1587655962 (1.27.4)
	Hombli HBEB-0224	com.hombli.smart (2.0.3)	1476038766 (2.0.3)
Functionalities: turn on/off			
Plug	CUCO EP2	com.gosund.smart (5.6.6)	1456965228 (5.6.6)
	Tapo P100M	com.tplink.iot (3.11.109)	1472718009 (3.11.105)
	Meross MSS425F	com.meross.meross (3.32.2)	1260842951 (3.33.0)
Functionalities: volume up/down			
Hub	EchoDot v5	com.amazon.dee.app (2.2.634135.0)	944011620 (2.2.672797)
Functionalities: turn on/off			
Other	Levoit Air Purifier	com.etekcity.vesyncplatform (5.5.61)	1289575311 (5.5.61)

Table 5.2: Software used on server and their versions.

Software	Version	Description
adb	1.0.41	Library for communicating with Android devices.
libimobiledevice	1.3.0-8.1build3	Library for communicating with iOS devices.
tcpdump	4.99.4	Command line tool to capture and analyse network traffic.
python	3.12.3	Programming language widely used for scripting and automation.

5.2 Android Phone Preparation

Next, we prepare the Google Pixel phone for the Android experiments. We summarize the software and versions used to setup the Android phone and to run the Android experiments in Table 5.3.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
dhcp-host=24:xx:xx:xx:xx:xx,iot0002,192.168.13.14,infinite
```

Listing 5.1: Example of *dnsmasq* entry.

Table 5.3: Software used for Android phone setup and Android experiments. Packages marked with * are installed on the server.

Software	Version	Description
Android	16	Operating system.
Build number	BP2A.250705.008	Unique identifier representing a specific Android software build.
PCAPDroid	1.8.7	Android app used to capture network traffic.
Magisk	29	Tool to root Android.
frida*	17.2.12	Dynamic instrumentation toolkit.
mitmproxy*	11.1.3	Interactive HTTPS proxy.

5.2.1 Firmware Flashing

We flash the Pixel 6a using the official Android Flash Tool, which offers an easy way to install factory images. We first navigate to the Android Flash Tool website at `https://flash.android.com/`. We then connect the Pixel 6a to the laptop via USB and select *Add New Device*, granting browser permissions to access the device. After selecting the Pixel 6a from the list, we scroll down to the *More Releases* section and choose the *Android 16* factory image. We ensure the *Unlock bootloader* option is selected. Unlocking is required so the tool can write protected system partitions during the flash. We then proceed by following the on-screen instructions to complete the installation.

5.2.2 Gaining Root Access

We require full administrative control over the device to perform in-depth instrumentation and traffic monitoring. We obtain root access by following a Magisk-based [65] rooting procedure, inspired by the guide from GetDroidTips [23]. The process involves enabling the Android Debug Bridge (*adb*), patching the firmware's boot image with *Magisk*, and flashing it back to the device.

Prerequisites. We install *adb* and *fastboot* on the laptop. These tools allow USB-based device communication and boot-level operations.

Enabling ADB on Phone. We enable Developer Options by navigating to *Settings* → *About phone* and tapping the *Build number* seven times. We then enable *USB debugging* via *Settings* → *System* → *Developer options*.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Installing Magisk. We download the *Magisk* app package from the official GitHub repository and install it directly on the phone.

Download Firmware. We download the official factory firmware corresponding to the device's build number from the Google Developers page [20]. It is critical to precisely match the firmware build number, as flashing an incompatible version could brick the phone.

Boot Image Patching. We extract the *boot.img* file from the firmware archive (*image-shiba-<build>.zip*) and push it to the phone's internal storage using *adb* (see Listing 5.2)

```
adb push ./boot.img /storage/emulated/0/Download/
```

Listing 5.2: Push Boot Image to Phone.

In the *Magisk* app, we select the option to "Install" → "Select and Patch a File", then choose the previously transferred *boot.img*. After patching, *Magisk* generates a new image file (e.g., *magisk_patched-<rnd-chars>.img*), which we pull from the phone (see Listing 5.3).

```
adb pull /storage/emulated/0/Download/magisk_patched-
↪  <rnd-chars>.img .
```

Listing 5.3: Pull Patched Boot Image.

Flashing the Patched Boot Image. Once patched, we reboot the phone into fastboot mode and flash the modified boot image (see Listing 5.4).

```
fastboot flash boot ./magisk_patched-<rnd-chars>.img
```

Listing 5.4: Flash Patched Boot Image via *fastboot*.

Verification. To confirm that we have root access, we open a shell via *adb* and check for elevated privileges (see Listing 5.5).

5.2.3 Installing Required Tools

After rooting the phone, we install all necessary tools required for instrumentation and app-level traffic capture.

Magisk Modules. We install two Magisk modules:

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
adb shell
shiba:/ $ su
shiba:/ # whoami
root
```

Listing 5.5: Verify root access via *adb* shell.

- *magisk-frida* [61]: enables *frida* instrumentation by integrating the *frida-server* as a *Magisk* module.
- *AlwaysTrustUserCerts* [37]: automatically configures Android to trust user-installed certificates for all apps.

MITM Certificate Installation. To intercept HTTPS traffic using *mitmproxy*, we first need to install the proxy's Certificate Authority (CA) on the phone so Android trusts the dynamically generated server certificate. Otherwise, the TLS handshake fails with certificate errors. We start *mitmproxy* on the local AP and connect the Pixel 6a to the AP's Wi-Fi network. We manually configure the phone to use the AP's IP address and the default *mitmproxy* port (8080) as its proxy. We then navigate to `http://mitm.it` in the phone's browser to download the Android-specific CA certificate. After downloading, we install this certificate via the phone settings by going to *Settings* → *Security* → *Install CA certificate* and selecting the downloaded file.

PCAPDroid. *PCAPdroid* [17] is an open-source app that allows traffic to be filtered by app package ID, thus filtering out background traffic generated by the OS. We benchmark its accuracy by capturing the traffic generated by one app, *com.hombli.smart*, associated with the *Hombli* light bulb, with and without using *PCAPdroid*. We then confirm that the traffic captured by *PCAPdroid* contains all the contacted endpoints. We notice that traffic below the IP level is not included (e.g., ARP requests). Because such traffic is out of scope, we deem *PCAPdroid* a valid tool.

5.3 iOS Phone Preparation

We now proceed to setting up the iOS phone for the experiments. Table 5.4 summarizes the software needed to prepare the iOS phone and to run the iOS experiments. We use an older version of *frida* since *objection* [53], which we use to disable SSL pinning, is incompatible with the latest *frida* version.

5.3.1 Gaining Root Access

Similarly to Android, in order to install the tools necessary for the experiments, we require root access on the iPhone. Therefore, we jailbreak the device using *palera1n* [41],

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Table 5.4: Software used for iOS phone setup and iOS experiments. Packages marked with * are installed on the server.

Software	Version	Description
iOS	16.11.7	Operating system.
Build number	20H360	Unique identifier representing a specific iOS software build.
WebDriverAgent	4.12.1	Used for controlling iOS devices remotely.
frida*	16.7.19	Dynamic instrumentation toolkit.
objection*	1.11.0	Runtime mobile exploration toolkit.
mitmproxy*	11.1.3	Interactive HTTPS proxy.

a jailbreak built on the Checkm8 exploit [48] that abuses a hardware vulnerability in the bootrom present in the Apple A8 processor (used in iPhone series 6, released in 2014), up to the Apple A11 processor (used in iPhone series 8, released in 2017).

Installing the jailbreak. We connect the phone to the laptop over USB. We unpack the jailbreak tool and run the command in Listing 5.6 to start the process. The tool places the device in Device Firmware Update (DFU) mode and prompts us to press specific buttons. We follow the terminal instructions. After the procedure completes, the phone reboots and the jailbreak is active.

```
./palera1n -f
```

Listing 5.6: Start jailbreak with *palera1n*.

5.3.2 Installing Required Tools

Next, we install the tools needed for instrumentation and automated UI control.

frida. We download the arm64 DEB package for *frida-server* from their website [19], copy it to */private/var/mobile* with *scp* through an *iproxy* tunnel, and install it with *dpkg -i*. We check if the *frida* server is running by using *ps aux | grep frida*. From the server we call `frida-ps -U` and receive the process list, showing that instrumentation is ready.

MITM Certificate Installation. The process to intercept HTTPS traffic on iOS using *mitmproxy* follows the Android procedure up to downloading the certificate. We begin by starting *mitmproxy* on the local AP and connecting the iPhone to the same Wi-Fi network. We then manually configure the iPhone's proxy settings, specifying the AP's IP address and the default *mitmproxy* port (8080). Next, we open Safari and navigate to `http://mitm.it`, downloading the iOS-specific CA certificate. On iOS, we install

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

this certificate by navigating to *Settings* → *General* → *VPN & Device Management* and selecting the downloaded profile. Finally, we enable full trust for the installed certificate by navigating to *Settings* → *General* → *About* → *Certificate Trust Settings* and toggling the switch corresponding to the *mitmproxy* CA.

WebDriverAgent. To automate the UI interaction, we compile WebDriverAgent [18] in Xcode 16.2 with a paid Apple developer account, sign it, and deploy it to the phone. After WebDriverAgent starts, we control the app through the Python client *facebook-wda* [40]. With this client, we simulate taps, capture screenshots, and start or stop apps, which gives us complete remote control during the experiments.

5.4 User Preparation

We create a fictitious user to ensure consistent inputs across apps and devices. We register a dedicated email address for this persona and record all PII that apps typically request (e.g., full name, email, phone number, postal address, and date of birth). We store these values in a JSON file inside the experiment repository and use them for every account setup.

Purpose for Analysis. We search the captured traffic for these PII values in the analysis step using simple string matching. We also generate common variants (lowercased, MD5-hashed, and Base64-encoded) and include them in the search set to account for variations. A single, well-defined persona simplifies the analysis and makes potential PII leakage easier to detect.

5.5 Experiment Flow

We configure CapIoT to run 10 *No Frida* and 10 *Frida* iterations. We connect the IoT device to the local AP in both network scenarios. For LAN experiments, we also connect the phone to the local AP. We connect the phone to the remote AP for WAN experiments while the IoT device remains on the local AP. We disable Bluetooth for the WAN scenario since we do not want the devices to talk locally via Bluetooth. Table 5.5 summarizes the outcome of the IoT experiments for the LAN and WAN scenarios on Android and iOS.

Each experiment follows the same sequence. We start by installing the companion app on the phone. If this is the first run for the device, we create an account, sign in, and populate every profile field with the persona information prepared earlier. Next, we pair the phone with the IoT device and verify that basic functions (e.g., switching the device on and off) work as expected. We then record the tap coordinates if this is the first time we test the device. After the *No Frida* and *Frida* iterations finish, we collect some metadata, including the device's MAC and IP address from the DHCP lease table. In addition, we extract the installed app binaries (APK on Android and IPA on iOS) with

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

	LAN	WAN	LAN	WAN
Teckin	●	●	●	●
Philips	●	●	◐	◐
Xiaomi	●	●	●	●
Wiz	●	◐	●	●
Hombli	●	●	●	●
CUCO	●	●	●	●
Tapo	●	◐	●	●
Meross	●	●	●	●
EchoDot v5	●	●	◐	◐
Levoit	●	●	●	●

Table 5.5: Summary of our Experiments. We report for each device if the experiment was successful without and with Frida (● = "Fully Successful", ◐ = "Frida Failed"). Frida failed means more than half of the Frida runs failed.

CapIoT, ensuring we can reinstall the exact versions in future experiments. More details on the experiments flow in Chapter 6.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 6

CapIoT

To streamline the experiment flow (recording coordinates once, replaying the same interactions across repeated runs, and collecting network traffic), we develop CapIoT, a semi-automated framework that simulates user interactions and records network traffic. We only need to record the tap coordinates once per device and can then reuse them for future runs. This chapter describes how to install CapIoT, how to configure and use it.

6.1 Overview

CapIoT is implemented in Python and follows a modular structure. It provides a command-line interface (CLI) for running experiments and recording coordinates. Each experiment contains multiple *iterations* and creates a timestamped folder hierarchy to store all collected artifacts, including network captures, *mitmproxy* output, SSL key logs, and Bluetooth logs. CapIoT provides three core functions: (1) recording tap coordinates, (2) extraction of companion apps from phones (APK on Android, IPA on iOS), and (3) running automated experiments.

Overall, CapIoT reduces manual effort and improves repeatability. It ensures that identical user interactions can be replayed across multiple runs, while producing a consistent set of network traces suitable for systematic analysis.

6.2 Installation

The complete source code and documentation of CapIoT are publicly available on GitHub [29]. The steps below describe how to install and set up the tool on Linux.

Prerequisites. CapIoT relies on several third-party tools, including *adb*, *tcpdump*, and *frida*. Chapter 5 details the required setup for both the server and the phones. Before starting, we ensure all binaries are installed and that the device-side `frida-server`

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

version matches the host `frida` package, since mismatched versions are a common cause of attach failures.

Clone the Repository. First, we clone the repository. Listing 6.1 shows the single command required to download the source code.

```
git clone https://github.com/SecPriv/capiot.git
```

Listing 6.1: Clone CapIoT repository from GitHub.

Create Separate Python Environments for Android and iOS. Android and iOS require different *frida* versions since *objection* [53], which we use to disable SSL pinning, is incompatible with the latest *frida* version. We define two dependency groups in *pyproject.toml* to avoid dependency conflicts and create a dedicated virtual environment for each platform. The environment named *.venv-android* holds the libraries we use on Android, while *.venv-ios* holds the iOS libraries. Listing 6.2 shows the commands we use to create the virtual environments.

```
python3 -m venv .venv-android
python3 -m venv .venv-ios
```

Listing 6.2: Create two virtual environments.

Install CapIoT. To install CapIoT and its dependencies for a given platform, we first activate the corresponding virtual environment, upgrade `pip`, and install the target dependency group (e.g., *.[android]* or *.[ios]*). We use *deactivate* to exit the environment before switching to the other environment. Listing 6.3 lists the commands required to install CapIoT and its platform-specific dependencies.

Verify Installation. Running the commands in Listing 6.4 confirms that the CLI loads and the provided configuration is valid.

6.3 Extract Apps from Phones

To ensure version consistency across LAN and WAN experiments, CapIoT can extract the app currently installed on the phone (APK on Android, IPA on iOS) and archive it. Dumping the app enables reinstalling the same binary in subsequent runs, improving reproducibility.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
# Activate Android env
source .venv-android/bin/activate
python -m pip install --upgrade pip
pip install -e ".[android]"
# capiot --version
# Switch away:
deactivate

# Activate iOS env
source .venv-ios/bin/activate
python -m pip install --upgrade pip
pip install -e ".[ios]"
# capiot --version
# Switch away:
deactivate
```

Listing 6.3: Install CapIoT and its platform-specific dependencies.

```
capiot --version
capiot config show -c examples/config.lan.android.yaml
```

Listing 6.4: Verify the CapIoT installation.

6.3.1 Android

On Android, CapIoT connects via *adb*, locates the app's APK, and pulls it to the host. Only the package name, phone ID, and output path are required.

```
capiot dump_app\
  --package-name com.hombli.smart \
  --phone-id 43409JAS90 \
  --output /home/example/apk
  --platform android
```

Listing 6.5: Dumping an Android APK from the phone.

6.3.2 iOS

On iOS, CapIoT retrieves the app bundle over SSH and packages it into an IPA.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
capiot dump_app\
  --package-name com.hombli.smart \
  --app-name Hombli
  --phone-id 43409JAS90 \
  --output /home/example/ipa
  --platform ios
  --ssh-port 2222
  --ssh-key ~/.ssh/iphone
```

Listing 6.6: Dumping iOS IPA from Phone.

6.4 Running Experiment

We start an experiment via the CLI by providing a configuration file, the app under test, the phone identifier, and an experiment label. Listing 6.7 shows an example of how to start an experiment.

```
capiot run \
  -c config/config.lan.android.yaml \
  --package-name com.hombli.smart \
  --phone-id 43409JAS90 \
  --device-name hombli_bulb
```

Listing 6.7: Starting an experiment with CapIoT.

6.4.1 Configuration

The framework supports two network profiles. We capture traffic on the phone and the local AP interface in the LAN profile. In WAN mode, we keep those local captures and, through an SSH tunnel, also record traffic on the remote access point to which the phone connects.

We manage configuration via YAML files, which specify parameters such as the platform (Android or iOS), network profile, interfaces, SSH credentials (for WAN), and output paths. We provide per-run arguments (e.g., the package name of the app under test, the phone ID, and the experiment label) on the CLI. At the same time, we fix all static parameters (e.g., interfaces, base output folder) in the YAML configuration. We read sleep times between actions (e.g., after the app started/stopped, between triggering taps) from another YAML file, which is hot-reloaded at runtime, allowing dynamically adjusting the sleep times without restarting the tool. The path to this YAML file can be specified using the “sleep_times_path” parameter in the experiment configuration. We

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

sometimes need to increase the wait time after launching the app during *Frida* iterations because instrumentation slows the startup. Listing 6.8 shows a sample configuration. The repository provides example configurations for each network profile and platform. Listing 6.9 shows an example configuration for Android using the LAN profile. Lines beginning with "#" indicate optional fields. We use default values if these parameters are omitted.

```
start_app: 15
stop_app: 10
after_tap: 2
after_similarity: 9
between_iterations: 300
```

Listing 6.8: Example of sleep time configuration. All values are in seconds and are hot-reloaded while the experiment is running.

```
network_profile: lan        # lan | wan
platform: android           # android | ios
server_interface: wlan0
phone_interface: wlan0
output_path: /home/example/Documents/capiot
frida_iterations: 10
no_frida_iterations: 10
tap_coordinates_path: /home/example/capiot/coordinates
image_crop_regions_path:
 ↪  /home/example/capiot/image_crop_regions.json

#sleep_times_path: /home/example/capiot/sleep_times.yaml

android:
  pcapdroid_api_key: xxxxxx
  pcap_download_path:
   ↪  /storage/emulated/0/Download/PCAPdroid/
  bluetooth_log_path:
   ↪  /data/misc/bluetooth/logs/btsnoop_hci.log
```

Listing 6.9: Example experiment configuration for Android with the LAN profile.

6.4.2 Android Experiment Flow

CapIoT automates the following experiment phases:

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

1. **Setup phase:** optional capture of initial device setup traffic.
2. **No-Frida phase:** repeated iterations where the app is launched and controlled without instrumentation.
3. **Frida phase:** repeated iterations where the app is launched and controlled with *frida* and *mitmproxy*.
4. **Teardown:** collection of Bluetooth logs, uninstall app, and device reboot.

Each experiment iteration falls into one of two categories: a run without instrumentation (*No Frida*) and an instrumented run with *frida* and *mitmproxy*. We execute 10 iterations for each category. The environment is not always stable since we work with rooted devices and use dynamic instrumentation. To automatically detect whether an experiment has failed, we capture a screenshot after each tap and compare it against a baseline image. To reduce false positives from dynamic UI elements (e.g., the clock, notifications, or rotating banners), we crop the screenshot to a region of interest and compute the similarity only within that region. We define this region once during coordinate recording. If the image similarity score falls below 99%, we flag the run as failed.

Initial Setup. We begin each run by installing the companion app and launching CapIoT. The tool starts an overall packet capture on the server to record network traffic and enables Bluetooth (only for the LAN scenario) on the phone. We then start the app. Next, CapIoT asks whether we want to record the setup. If we answer yes, the tool opens a separate PCAP file and pauses execution while we finish the setup. During this pause, we create or log in to an account, and fill out every PII field in the profile using the persona data from Section 5.4. We then pair the phone with the IoT device and confirm that the basic functions work correctly. We confirm that the setup is completed by pressing "y" and CapIoT moves on to the next step.

Recording Coordinates. In the next step, CapIoT asks whether we need to record tap coordinates. If we have them already, we skip ahead. Otherwise, we follow the routine in Section 6.5. While recording the coordinates, the tool pauses again.

Defining Crop Regions. To make the screenshot comparison more robust, we crop each image to the area that changes after a tap. We open the baseline screenshots in GIMP, note the x and y offset plus width and height of the region of interest, and add these numbers to a JSON file. Each entry in the array matches one recorded tap. After saving the file, we press "y" so CapIoT can read the crop definitions and begin the first *No Frida* run. The array name must match the value we passed with the *–device-name* parameter when we started the experiment. Listing 6.10 shows an example.

No Frida Run. For the experiment iterations without *frida*, we first launch *PCAPdroid* on the phone to capture traffic. Concurrently, we start *tcpdump* on the server to record network traffic passing through the AP. We then open the companion app on the phone and trigger the GUI interactions. We read coordinates from the text file for each

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
{
  "hombli_bulb": [
    {
      "x": 906,
      "y": 501,
      "width": 75,
      "height": 75
    },
    {
      "x": 906,
      "y": 501,
      "width": 75,
      "height": 75
    }
  ]
}
```

Listing 6.10: Example crop region file used by CapIoT.

interaction, simulate a tap, make a screenshot, and compare it to the previously recorded baseline image. If the similarity score is below 99%, we mark the attempt as failed. After completing all interactions, we wait five minutes to capture potential background traffic. Afterward, we stop *PCAPdroid* and *tcpdump*, transfer the PCAP file from the phone to the server, delete the local PCAP file from the phone, and terminate the app.

Frida Run. When running experiment iterations involving *frida* instrumentation, we first set up traffic redirection with *iptables*, the Linux firewall/NAT tool. We keep DNS and DHCP (ports 53 and 67) untouched, and redirect the phone's other traffic to *mitmproxy* on the server. Listing 6.11 shows the rules that we apply. We then start *PCAPdroid* on the phone and *tcpdump* on the server to capture network traffic. Next, we run *mitmdump* in *transparent* mode, meaning the phone does not need an explicit proxy setting since all packets are redirected to the proxy by the *iptables* rules. We save all flows to a file and export TLS session keys so we can decrypt captures in Wireshark later. In addition, we run a custom script *communication/mqtt_message.py* to parse MQTT packets (e.g., *CONNECT*, *SUBSCRIBE*, *PUBLISH*) and log topics and payloads to the *mitmproxy* log. Furthermore, we accept all redirected TCP/UDP connections (*–tcp-hosts ".*"* and *–set udp_hosts=".*"*), and disable upstream certificate checks (*–ssl-insecure*). Listing 6.12 shows the command we use.

Then, we start the *frida-server* with an SSL unpinning script [1], which automatically starts the app. Similar to the *No Frida* run, we perform GUI interactions by reading coordinates from the previously created file, triggering taps, taking screenshots, and

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
sudo iptables -t nat -I PREROUTING -i <interface> -p udp
 ↪  --dport 53 -j ACCEPT
sudo iptables -t nat -I PREROUTING -i <interface> -p tcp
 ↪  --dport 53 -j ACCEPT
sudo iptables -t nat -I PREROUTING -i <interface> -p udp
 ↪  --dport 67 -j ACCEPT
sudo iptables -t nat -A PREROUTING -i <interface> -p tcp -s
 ↪  192.168.13.14 -j REDIRECT --to-port 8080
sudo iptables -t nat -A PREROUTING -i <interface> -p udp -s
 ↪  192.168.13.14 -j REDIRECT --to-port 8080
```

Listing 6.11: *iptables* rules applied for traffic redirection.

```
export SSLKEYLOGFILE=<sslkeys_dumps_path>
mitmdump --mode transparent -p 8080 -w <mitm_dumps_path>
 ↪  --tcp-hosts \".\" --set udp_hosts=\".\" --ssl-insecure -s
 ↪  communication/mqtt_message.py
```

Listing 6.12: Mitmdump command.

comparing them with baseline images. If the image similarity score is below 99%, we mark the run as failed. After waiting five minutes, we stop *PCAPdroid* and *tcpdump*, transfer the captured PCAP file from the phone to the server, and delete the local copy on the phone. We terminate *frida*, *mitmdump*, and the companion app. Finally, we remove all previously applied *iptables* rules.

Post-Run Cleanup. After both the *No Frida* and *Frida* runs, we retrieve the Bluetooth log file from the phone and delete it on the phone. We then uninstall the companion app and reboot the phone to reset the Bluetooth logging system for the following experiments.

Additional Metadata Collection. In addition to running the experiment, we collect metadata essential for reproducibility and analysis. We retrieve the MAC and IP addresses of the tested IoT device from the DHCP lease file and extract the companion mobile apps (APK for Android, IPA for iOS) from the phones using CapIoT.

6.4.3 iOS Experiment Flow

The experiment flow on iOS largely mirrors the Android setup, with a few platform-specific adjustments due to system restrictions and tooling limitations.

Traffic Capture. Since there is no iOS equivalent to *PCAPdroid*, we capture network traffic directly on the iPhone using *tcpdump*. Additionally, we enable the *App Privacy*

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Report feature in the iOS settings, which logs the domains contacted by each app. To ensure clean logs for each run, we disable and re-enable this setting before the start of every experiment. We manually record the contacted domains before uninstalling the app, as the report is cleared upon removal.

Bluetooth Logging. We do not collect Bluetooth logs on iOS because we cannot retrieve them without a macOS device. Consequently, we omit the reboot step that is otherwise required on Android to regenerate the log file. Rebooting the iPhone would also break the jailbreak, which would require applying the jailbreak again.

frida. To disable SSL pinning on iOS, we use *objection*, a runtime mobile exploration tool that uses *frida* in the background. We found no reliable *frida*-based script for generic SSL unpinning for iOS, and therefore rely on *objection* to perform this step.

6.5 Recording Tap Coordinates

To automate UI interaction, we need to get the coordinates of the UI elements we want to tap. *CapIoT* provides a recording mode for this purpose. Listing 6.13 shows how to start recording coordinates. The tool walks us through the process with on-screen prompts. On Android, we tap the desired element and *CapIoT* automatically extracts the corresponding screen coordinates. On iOS, we need to perform additional manual steps.

```
capiot record_coordinates \
  --package-name com.hombli.smart \
  --phone-id 43409JAS90 \
  --device-name hombli_bulb
  --output /home/example/coordinates_android
  --platform android
```

Listing 6.13: Recording tap coordinates with CapIoT.

6.5.1 Android

We manually perform taps on the screen to trigger the relevant actions (e.g., turning the smart bulb on/off). After each tap, CapIoT automatically logs the tap coordinates and makes a screenshot. When we finish, we press `Ctrl+C`. The captured screenshots and coordinates are stored in the output directory we passed on the command line. The output includes screenshots, which will be the baseline, and a text file containing the tap coordinates. Listing 6.14 shows an example.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
tap 964 480
tap 946 502
```

Listing 6.14: Tap list produced by CapIoT for Android.

6.5.2 iOS

Unlike Android, iOS does not provide a mechanism to capture tap events in real time. As a result, we follow a more manual procedure to collect the required coordinates. CapIoT prompts us to press "y" to capture a screenshot. We perform the required tap on the phone's screen and then press "y" to make a screenshot. We repeat this procedure for each GUI interaction. Once we have completed all interactions, we stop recording by pressing *Ctrl + C*. We then open each captured screenshot in GIMP, identify the relevant tap locations by left-clicking, and manually write the coordinates into the coordinates text file, using the same format as Android (see Listing 6.14).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 7

LAN vs. WAN

In this chapter, we analyze how IoT devices and their companion apps behave under different network scenarios by comparing communication when IoT devices and phones are connected to the same network (LAN) and when IoT devices and phones are connected to different networks (WAN). We analyze and compare multiple aspects of communication, including endpoints, tracking services, cloud providers, country distribution, and protocol usage, to highlight differences in security, privacy, and compliance across network settings and mobile platforms.

7.1 Endpoint Analysis

7.1.1 Android

For Android, we observe clear differences in the number of endpoints contacted between LAN and WAN scenarios. Device communication in the WAN involves slightly fewer unique endpoints than the LAN. For example, across all devices, Android devices contact 72 endpoints in WAN and 81 in LAN. We explain this LAN increase by including local IP addresses, which appear when the device and phone communicate directly over the LAN. On the app side, a similar pattern occurs: WAN traffic involves slightly fewer endpoints than LAN (155 vs. 167). Again, the additional LAN endpoints reflect local IPs and service interactions, which are unnecessary when all traffic goes through the cloud in WAN mode. We show our results in Table 7.1.

We show a significant difference when comparing devices vs. apps within Android. The app consistently contacts a much larger number of endpoints than the device, both in WAN (155 vs. 72) and LAN (167 vs. 81). This implies that companion apps shoulder much of the communication overhead, connecting not only to vendor clouds but also to analytics, advertising, and platform-specific services (e.g., Google APIs, content delivery networks).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

7.1.2 iOS

For iOS, we also show variations between LAN and WAN, though the pattern differs slightly from Android. On the device side, iOS endpoints decrease when moving from WAN (75 endpoints) to LAN (72 endpoints). This contrasts with Android, where the LAN count is higher. One reason for the increase is that *Philips* contributes significantly by introducing multiple cloud endpoints in WAN mode that we do not observe in LAN. The *Philips* device connects to a wide set of HealthSuite domains (e.g., `firmware.eu01.iot.hsdp.io`, `profile.eu01.iot.hsdp.io`) in WAN mode, making the overall count in WAN mode higher than in LAN.

On the app side, we see the opposite trend: endpoint counts increase from 124 in WAN to 133 in LAN. As with Android, more local IPs and discovery services explain this increase, as LAN connectivity enables direct phone–device communication in addition to cloud-based interactions.

When comparing the device versus the app within iOS, the app again contacts substantially more endpoints than the device. The disparity is evident in both WAN (124 vs. 75) and LAN (133 vs. 72). As with Android, this reflects the role of the mobile application as the primary interface for device management, cloud services, and third-party integrations. In contrast, the device communicates with a more focused set of vendor-controlled endpoints.

7.1.3 Comparison

A cross-platform comparison reveals both similarities and differences in communication behavior. In both ecosystems, companion apps consistently generate more endpoint connections than devices, underscoring their central role in ecosystem integration. However, Android phones generally contact more endpoints than their iOS counterparts, particularly in WAN (155 vs. 124) and LAN (167 vs. 133), although the number of shared endpoints remains similar (65 for WAN, 69 for LAN). This indicates that while Android apps connect to a broader set of endpoints overall, the common subset of vendor and cloud servers accessed by both platforms remains relatively stable across network scenarios. The higher endpoints count in Android is driven by Android's heavier reliance on Google services (e.g., `android.googleapis.com`, `playgateway-pa.googleapis.com`, `people-pa.googleapis.com`), as well as additional analytics and advertising endpoints. By contrast, iOS apps show a somewhat leaner set of endpoints, including Apple ecosystem services and vendor-specific domains not observed on Android.

Overall, both platforms rely on a mixture of cloud and local endpoints. Still, Android generates a broader and more diverse communication footprint at the app level, with strong integration into Google's ecosystem and third-party services. iOS applications, while also diverse, appear more consolidated and include Apple ecosystem–specific endpoints. At the device level, both ecosystems behave similarly, with communication primarily directed at vendor-controlled servers and only modest changes between LAN and WAN conditions.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

	Device Traffic						**App Traffic**					
	WAN			LAN			WAN			LAN		
Teckin	6	6	4	7	6	4	26	10	6	27	12	8
Philips	31	35	22	35	26	15	31	25	14	34	20	14
Xiaomi	2	2	1	3	3	1	15	24	8	16	28	6
Wiz	3	3	1	4	4	1	8	7	4	6	7	4
Hombli	3	6	2	5	4	2	8	5	3	9	6	4
CUCO	3	3	1	4	4	1	4	9	2	7	11	2
Levoit	3	3	2	3	3	2	12	6	4	12	4	4
Meross	4	4	3	5	5	3	16	4	2	19	8	3
EchoDot v5	14	11	10	11	13	9	22	23	16	24	23	16
Tapo	3	2	1	4	4	2	13	11	6	13	14	8
Total	72	75	47	81	72	40	155	124	65	167	133	69

Table 7.1: Number of unique endpoints contacted by IoT devices and their companion apps across different network scenarios. The table distinguishes between **Device Traffic** (left) and **App Traffic** (right), each measured under WAN and LAN conditions. Counts are reported separately for Android (), iOS (), and their shared endpoints ().

7.2 Tracking Endpoints Analysis

We cannot find any tracking endpoints contacted by devices; thus, we focus on the ones contacted by the companion apps. We show notable differences across platforms and network conditions.

7.2.1 Android

Android apps show quite some tracking activity. Four apps contact at least one tracking endpoint. On the WAN, Android apps contact on average 2.80 tracking endpoints per app, with a high standard deviation of 2.68, and up to seven endpoints in a single app (*Teckin*). In the LAN scenario, the number of tracking endpoints surprisingly increases, reaching an average of 3.50 per app. The standard deviation is 2.65, again with a maximum of seven endpoints for the same app.

7.2.2 iOS

For iOS applications, the average number of tracking endpoints per app remains relatively low. However, seven apps in this case contact at least one tracking endpoint, significantly a higher number than Android. When operating on the LAN, iOS apps contact on

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

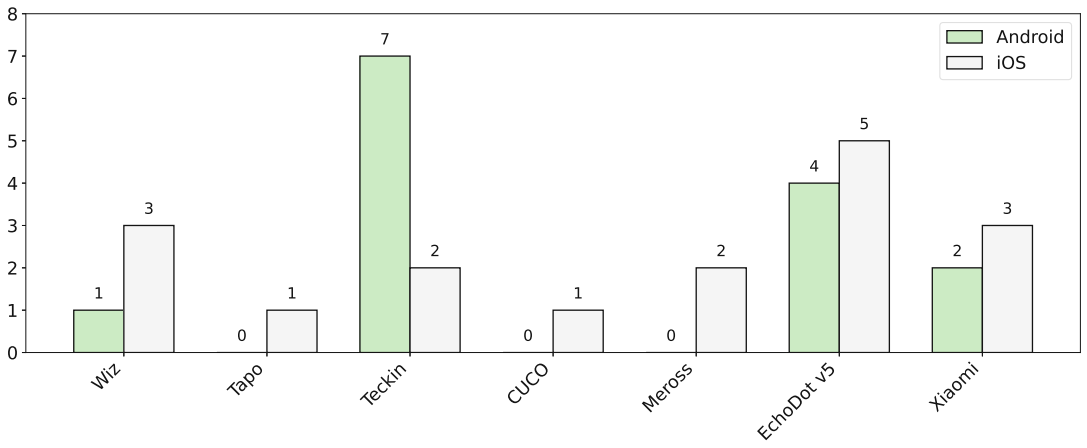


Figure 7.1: Comparison of the number of distinct third-party tracking endpoints contacted by each companion app when run on Android versus iOS. Only apps with at least one tracker on either platform are shown.

average 2.43 tracking endpoints, with a standard deviation of 1.40, and a maximum of five tracking endpoints for any single app (*EchoDot v5*). In the WAN scenario, the average decreases slightly to 2.00 tracking endpoints, although the standard deviation increases to 1.53, again with a maximum of five endpoints. This suggests that iOS applications maintain a relatively consistent number of tracking connections regardless of network configuration, with some variability among individual apps. These results indicate a greater prevalence of tracking on iOS compared to Android and heterogeneity among Android apps.

7.2.3 Comparison

We show that iOS companion applications pose a higher risk of third-party tracking than their Android counterparts, both in terms of the average number of tracking endpoints and in the variability across applications. From a security and privacy perspective, more tracking endpoints imply a wider leakage of potentially sensitive data, including device usage patterns, user behavior, and contextual information. We report our results in Figure 7.1.

7.2.4 Country Distribution of Tracking Endpoints

The country distribution of tracking endpoints highlights significant differences between Android and iOS in the WAN and LAN scenarios. We report our results in Figure 7.2. For Android, tracking endpoints are primarily concentrated in Austria (AT), the Czech Republic (CZ), Ireland (IE), the United States (US), and the Netherlands (NL). In the WAN scenario, Android apps communicate most frequently with tracking endpoints in the Czech Republic and the US, followed by Austria and Ireland, with limited connections

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

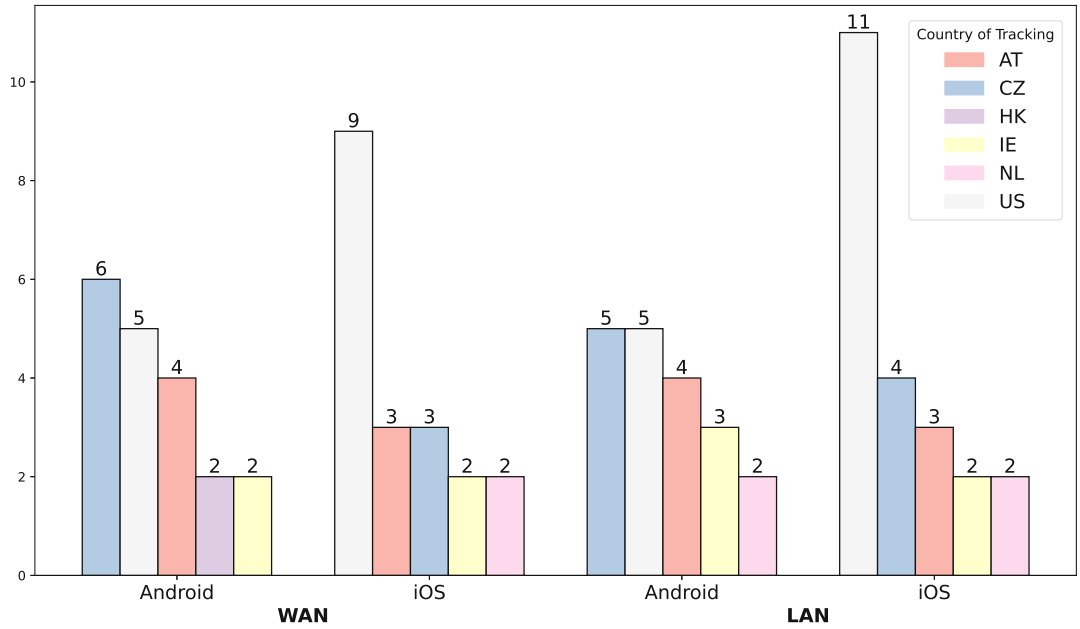


Figure 7.2: Country distribution of tracking endpoints in apps' traffic for the different OS and network scenarios.

to Hong Kong. Android apps show a similar geographic distribution in the LAN setting, though with slightly higher engagement with Czech and Irish endpoints, and consistent connections to the Netherlands. In contrast, iOS phones show a stronger dependence on US-based tracking endpoints across both scenarios, with nine endpoints in the WAN scenario and eleven in the LAN one. Austrian, Czech, Irish, and Dutch trackers also appear regularly, although in lower numbers than those from the United States. We show that US-based tracking endpoints are predominant in iOS apps regardless of network conditions, whereas Android apps distribute their tracking endpoints more across European regions. From a privacy perspective, the stronger concentration of iOS tracking traffic in the United States raises potential concerns regarding compliance with European data protection regulations, particularly the GDPR.

7.2.5 Event-Based Tracking

To understand if a correlation exist between action triggered from the devices and amount of data sent to tracking endpoints, we compute Cumulative Distribution Function (CDF) plots correlating time with the number of bytes sent to *Tracking & Analytics* endpoints. For both Android and iOS in two apps (i.e., *Teckin* and *EchoDot v5*) for the WAN scenario, the amount of traffic to these endpoints grows fast while we interact with the device/app but then reaches a plateau when our interaction ends. Interestingly, while the same pattern is found for Android in the LAN setting, in iOS none of the apps exhibit such behavior. We provide an example in Figure 7.3, where we see that the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

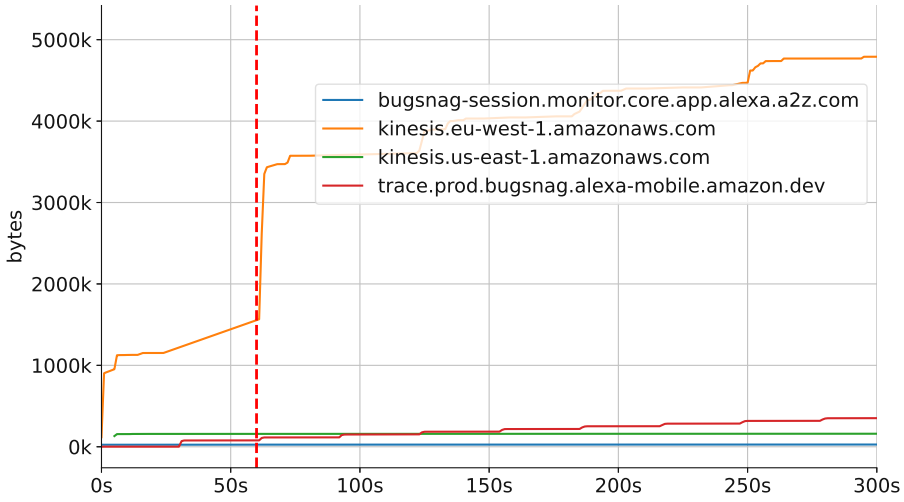
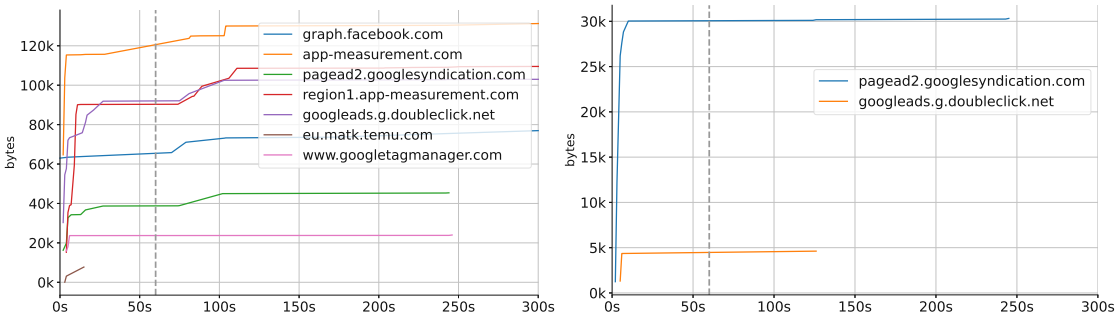


Figure 7.3: Cumulative traffic volume of the *EchoDot v5* toward analytics endpoints. Most traffic occurs before the 60s mark (red dashed line), i.e., during user interaction with the device, after which traffic stabilize and reach a plateau. This indicates that analytics-related communication is triggered based on user actions.

tracking traffic for the *EchoDot v5* in the LAN scenario for Android grows fast during our interaction, i.e., before the red vertical dotted line at 60s, and then abruptly stops. At the same time, we also show how no pattern can be found for *Teckin* for iOS in the LAN scenario, compared to Android in Figure 7.4. By investigating the analyzable traffic we captured with *frida* and *mitmproxy*, we could identify what we refer to as *event-based* tracking. With that we mean instances where we found payload values in the traffic that we associated with the activity we performed on the app at that specific moment, e.g., turning a device on/off. Some concrete examples are *"interactionType":"click"* or *"appComponent":"Favorites"* that we find in *EchoDot v5* and signalling that the user clicked on the Favorites button in the app UI. To confirm our findings, we performed additional experiments without any interaction and recorded traffic for one hour on each device, where we initially found event-based tracking. Indeed, these second captures confirm that tracking endpoints receive traffic mostly when actively interacting with the app and that *event-based* payloads only appear during that stage.

Clearly, such form of analytics and tracking poses significant risks to users' privacy. Services receiving such information are capable of perfectly defining users' interaction behavior, thus inferring critical information to better tailor content and advert to them. Further, the *Teckin* app relies on third-party services for this. Vendors, and consequently their customers, outsource their trust in how data is handled, stored, and processed to such third-party services, posing serious regulatory and security issues.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.



(a) LAN scenario for Android.

(b) LAN scenario for iOS.

Figure 7.4: *Teckin* cumulative traffic volume towards Tracking & Analytics endpoints.

7.3 Local Communication

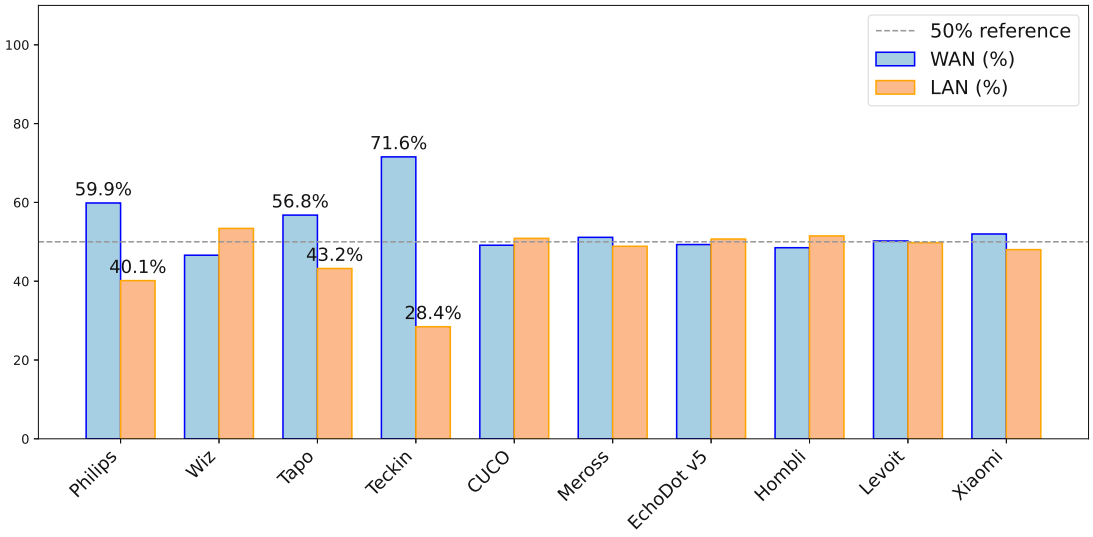
7.3.1 Android

For Android apps, the WAN/LAN distribution of traffic directed to remote endpoints varied considerably across vendors. *Philips* and *Tapo*'s apps communicate more to remote endpoints in the WAN scenario (59.9% and 56.8%, respectively), with *Teckin* exhibiting an even stronger bias (71.6%). Conversely, all the remaining apps show balanced traffic close to 50%. Showing the same amount of traffic to remote endpoints in both WAN and LAN scenarios suggests that vendors do not follow ENISA guidelines and do not prefer local communication when possible. We see the same trend for the devices for *Philips* and *Hombli*, with *Teckin* showing basically only remote traffic in the WAN scenario. This indicates that in this case, the device chooses local communication when connected to the same network as the phone. Instead, *Tapo* shows more traffic to remote endpoints in the LAN scenario. We show our results in Figure 7.5.

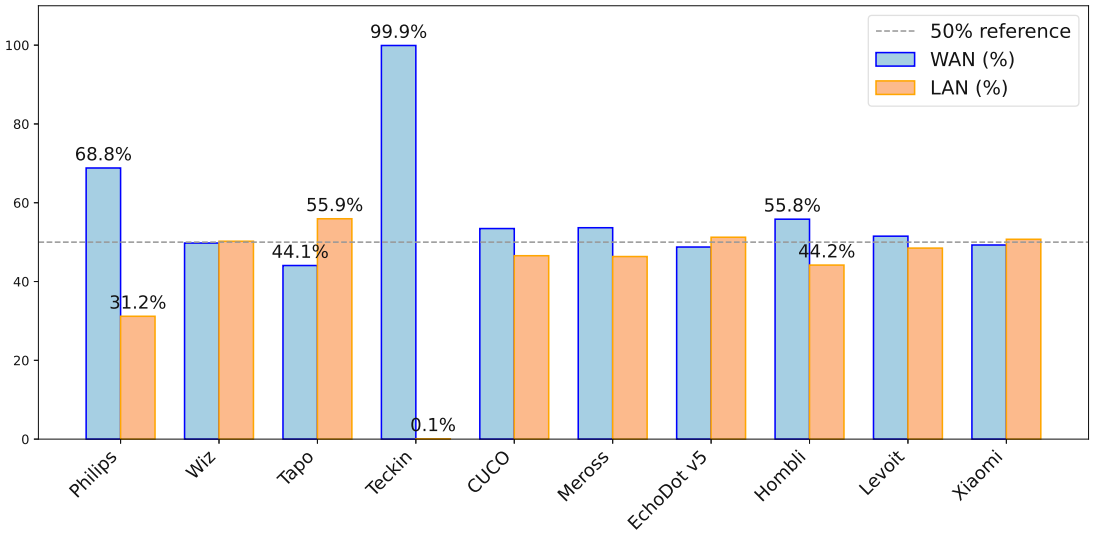
7.3.2 iOS

On iOS, the WAN/LAN remote traffic ratio is far more polarized. At the device level, *Philips* and *Teckin* transmit almost exclusively to remote endpoints over the WAN (99.7% and 99.9%, respectively), with LAN remote traffic accounting for less than 1%. In contrast, *EchoDot v5* shares more traffic with remote endpoints in the LAN (80.0%). Overall, four devices show less traffic to remote locations in the LAN scenario, and the remaining five show more or less the same amount. The distribution is less extreme at the app level but still reveals strong preferences. *Philips* and *Teckin* again favor remote communication in the WAN setting (83.2% and 93.8%, respectively). *Wiz*, *Meross*, *EchoDot v5*, and *Xiaomi* lean toward more remote traffic in the LAN scenario (56.6%, 57.3%, 59.7%, and 56.1% respectively). We show our results in Figure 7.6.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN



(a) **Android** - WAN vs LAN Traffic Ratio per App



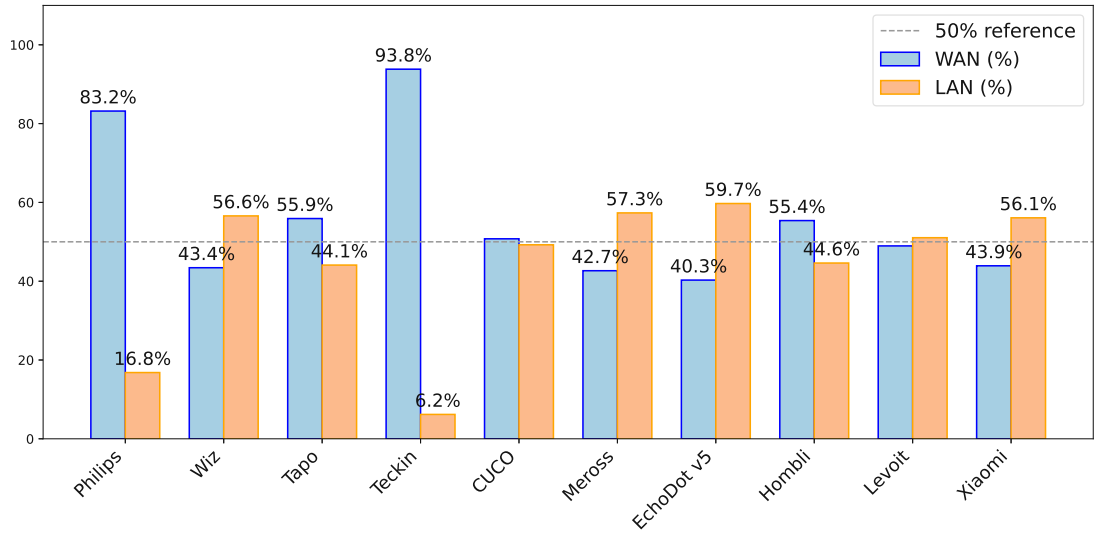
(b) **Android** - WAN vs LAN Traffic Ratio per Device

Figure 7.5: Comparison of WAN vs. LAN traffic ratios for Android. We highlight in gray the 50% mark, i.e., the app or device shares the same amount of traffic to remote endpoints in both the WAN and LAN scenario.

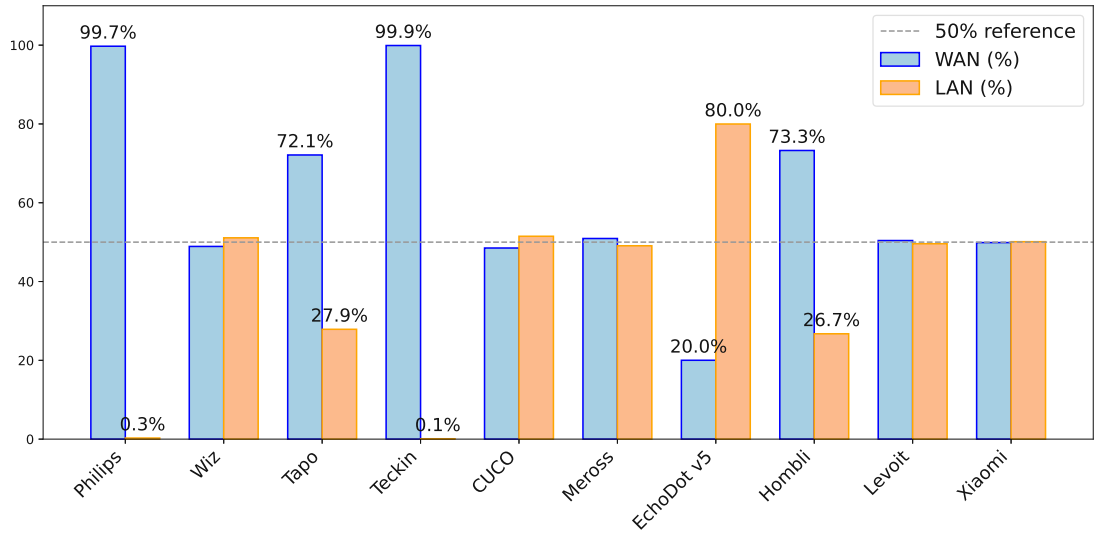
7.3.3 Comparison

Across ecosystems, a consistent pattern emerges: Android distributes its remote traffic more evenly between WAN and LAN scenarios, with several vendors operating close to the 50% threshold. By contrast, iOS adopts more extreme communication strategies,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.



(a) **iOS** - WAN vs LAN Traffic Ratio per App



(b) **iOS** - WAN vs LAN Traffic Ratio per Device

Figure 7.6: Comparison of WAN vs. LAN traffic ratios for iOS. We highlight in gray the 50% mark, i.e., the app or device shares the same amount of traffic to remote endpoints in both the WAN and LAN scenario.

either almost entirely communicating remotely in the WAN scenario or showing more remote traffic in the LAN scenario, where local communication should be preferred. This polarization is evident at the device level, where iOS traffic rarely approaches the 50% reference line.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Teckin	✔	✔
Philips	✔	✔
Xiaomi	✖	✖
Wiz	✔	✔
Hombli	✔	✔
CUCO	✔	✔
Levoit	✖	✖
Meross	✔	✔
EchoDot v5	✖	✖
Tapo	✔	✔

Table 7.2: Overview of IoT devices that use local communication in the LAN scenario. A checkmark (✔) indicates that the device communicates locally with the companion app (Android or iOS) when both are on the same network, while a cross (✖) indicates that all communication is routed via cloud services.

Device Traffic				**App Traffic**			
WAN		LAN		WAN		LAN	
AWS (59)	AWS (65)	AWS (64)	AWS (52)	AWS (83)	AWS (87)	AWS (92)	AWS (88)
Other (41)	Other (42)	Other (49)	Other (43)	Other (60)	Other (45)	Other (67)	Other (46)
Azure (3)	Azure (3)	Azure (3)	Azure (3)	Google (33)	Azure (18)	Google (31)	Google (13)
Oracle (2)	Oracle (2)	Oracle (2)	Oracle (2)	Azure (14)	Google (6)	Azure (10)	Azure (10)
–	–	–	–	Cloudflare (4)	Cloudflare (3)	Cloudflare (4)	Cloudflare (4)

Table 7.3: Top five cloud providers observed in WAN and LAN communications, broken down by device and app traffic, for both Android and iOS. Counts in parentheses indicate the number of endpoints associated with each provider. We report as *Other* the providers that we could not identify.

In addition, Table 7.2 shows that most devices (e.g., *Teckin*, *Philips*, *Wiz*, *Hombli*, *CUCO*, *Meross*, and *Tapo*) do establish local communication in LAN. However, some devices, such as *Xiaomi*, *Levoit*, and *EchoDot v5*, rely exclusively on cloud services even when local connectivity would be possible. The results are the same across Android and iOS. This behavior goes against ENISA guidelines, which recommend preferring local communication whenever feasible to minimize exposure to external networks and reduce security and privacy risks.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

7.4 Cloud Provider Analysis

7.4.1 Android

Amazon Web Services (AWS) is the predominant provider for Android across both WAN and LAN communications. In the WAN and LAN scenarios, app traffic is primarily directed to AWS, followed by "Other" providers and Google. Azure and Cloudflare account for only a minor share. LAN and WAN device traffic exhibits a similar pattern, with AWS and "Other" providers dominating, while Azure and Oracle appear only marginally.

7.4.2 iOS

For iOS, AWS also represents the primary provider in both WAN and LAN scenarios. WAN and LAN app traffic is mostly associated with AWS and "Other" providers, with smaller contributions from Azure, Google, and Cloudflare. In LAN and WAN device traffic, AWS and "Other" providers are again most prevalent, whereas Azure and Oracle contribute minimally.

7.4.3 Comparison

Overall, both ecosystems strongly rely on AWS as the dominant cloud service provider across WAN and LAN communication. We report our results in Table 7.3. However, notable differences emerge: Android applications show a stronger connection with Google services, whereas iOS traffic is comparatively more distributed, with Azure assuming a more prominent role in WAN communication. Cloudflare appeared sporadically in both ecosystems, with slightly higher activity in Android WAN traffic. LAN traffic in both ecosystems reveal limited diversity, almost exclusively split between AWS and "Other" providers.

7.5 Category Analysis

7.5.1 Android

In Android, endpoint categorizations are dominated by support services and first-party servers. In apps, a substantial share of endpoints fall within the support category (56–59), followed closely by first-party endpoints (44–47). Analytics and tracking services appeared with a consistent but smaller presence (14 in both scenarios). At the device level, traffic was directed mainly to first-party servers (26 in both scenarios), while support endpoints represented a more limited set (8–10). The limited but constant presence of analytics and tracker endpoints is notable from a security and privacy perspective. Although fewer than support and first-party endpoints, they can leak behavioral information and usage patterns outside the user's control. Conversely, the reliance on first-party endpoints in device communication suggests that many operations are confined within vendor-controlled infrastructures, which may mitigate some risks.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

	Device Traffic				App Traffic			
	WAN		LAN		WAN		LAN	
First	26(36%)	24(32%)	26(32%)	18(25%)	44(28%)	45(36%)	47(28%)	45(34%)
Support	8(11%)	9(12%)	10(12%)	9(13%)	56(36%)	31(25%)	59(35%)	34(26%)
Analytics	–	–	–	–	14(9%)	14(11%)	14(8%)	17(13%)
Remote IPs	28	31	27	27	25	31	27	30
Local IPs	10	11	18	18	16	3	20	7

Table 7.4: The endpoints categories observed in WAN and LAN communications, broken down by device and app traffic, for both Android ( ) and iOS ( ). For each category we report the absolute number of endpoints as well as their relative share of all endpoints in parentheses. We also include the number of remote and local IPs that we could not classify.

7.5.2 iOS

For iOS, the distribution between endpoint categories is more balanced, but with similar trends. App communication predominantly involves first-party endpoints (45 in both scenarios), accompanied by a considerable number of support-related endpoints (31–34) and analytics/tracking services (14–17). At the device level, first-party endpoints again represent the majority (18–24), with a smaller contribution from support endpoints (9 in both scenarios). The presence of analytics and tracking services in iOS traffic, slightly higher than in Android in the LAN scenario, carries similar privacy risks, as these services may collect telemetry data across applications. Although iOS devices use fewer support endpoints than Android, third-party trackers' presence underscores a risk of exposing sensitive identifiers or metadata beyond the vendor's infrastructure.

7.5.3 Comparison

Across both ecosystems, first-party endpoints clearly dominate device communications, while app traffic introduces greater variety through support and tracking services. In Android apps, support endpoints make up the largest share (∼36%), surpassing first-party endpoints (∼28%). In iOS apps, the situation is reversed, with first-party endpoints dominating (∼34-36%) and support services taking a smaller role (25%). We report all our results in Table 7.4. Analytics and tracking endpoints appear in both ecosystems with similar prevalence, although iOS shows a slightly higher share (up to 13% vs. 9% on Android). A similar amount of analytics and tracker endpoints across Android and iOS indicates that both platforms include monitoring components within their ecosystems, potentially harming user privacy. Dependence on support endpoints is not intrinsically problematic from a security standpoint, but it broadens the attack surface by distributing trust across multiple service providers. The privacy implications are worse concerning analytics and tracking endpoints, as these connections may expose behavioral data,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

device identifiers, and network attributes, often with limited transparency or user consent. Overall, while Android and iOS rely heavily on first-party infrastructures for device functionality, analytics and tracker endpoints highlight a problem between service support and user privacy.

7.6 Country Analysis

We perform all our experiments within the EU borders and having set an EU country in all app's preferences (when available). We investigate where the traffic is directed to understand if potential privacy issues might arise. The GDPR protects European citizens, and traffic directed to countries with a more lax data protection regulation might threaten users' privacy. We report all our results in Figure 7.7.

7.6.1 Android

For Android, companion apps and IoT devices primarily contact endpoints located in Germany, followed by Austria, Ireland, and, to a lesser extent, the Netherlands. These destinations are all within the European Union (EU), which suggests that most traffic remains under the jurisdiction of EU data protection regulations such as the GDPR. When the companion application and the device operate in the WAN scenario, the app communicates primarily with German servers (64 endpoints) and destinations in the Czech Republic, the United States, Hong Kong, Canada, and Singapore. This increase in endpoints outside the EU indicates that, although the IoT device communicates predominantly within the EU (Germany: 55 endpoints, Austria: 15, Ireland: 5), the companion app introduces significant cross-border traffic. We observe a similar pattern when the two entities operate in the LAN scenario. While the IoT device continues to concentrate its endpoints within the EU (Germany: 60, Austria: 17, Ireland: 5), the app sustains notable communication with non-EU endpoints, including the United States, Singapore, Hong Kong, and Canada. IoT devices maintain a primarily EU-centric communication pattern, whereas their companion applications extend data flows to non-EU countries. This difference increases the likelihood that personal data may leave the EU, creating potential compliance risks under GDPR.

7.6.2 iOS

In the case of iOS, endpoints also concentrate heavily in Germany, but the data reveals a more central role for non-EU destinations, particularly China. When the phone and the device are in the WAN scenario, the app contacts German servers (59 endpoints) most frequently, but also shows substantial interaction with Chinese (24), United States (22), and Singaporean (6) servers. The corresponding IoT device endpoints are mainly located in EU countries (Germany: 62, Austria: 17, Ireland: 5), with limited contact with the United States (7). In the LAN scenario, the app again demonstrates a dual pattern of communication, combining many endpoints in Germany (53) with considerable traffic to China (36), the United States (27), and Singapore (6). The device, by contrast,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

remains focused primarily on EU endpoints, with Germany accounting for 54 connections, in addition to limited endpoints in Austria, Ireland, and the United States. These results indicate that while IoT devices share the EU-centric communication profile, as for Android, iOS companion apps exhibit stronger engagement with non-EU endpoints, particularly in China. This raises concerns regarding data transfers to jurisdictions with weaker privacy protections.

7.6.3 Comparison

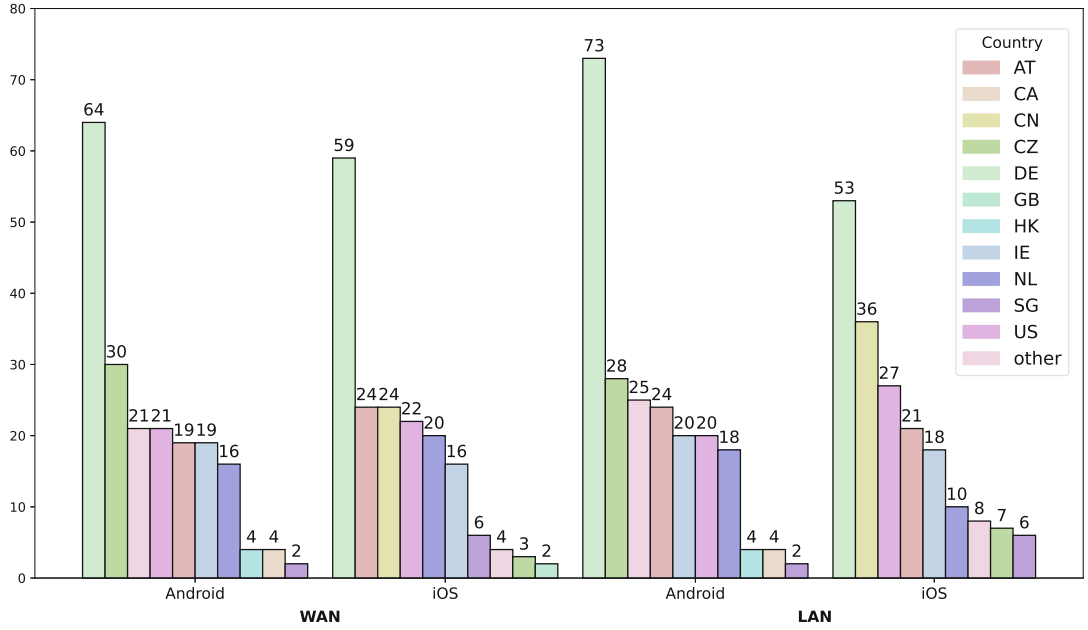
When comparing Android and iOS, we highlight that IoT devices predominantly communicate with servers located within the EU in both cases, especially in Germany. This EU-centric behavior limits cross-border data transfers and reduces the potential for conflicts with EU data protection requirements. The companion apps, however, exhibit different patterns. Android apps introduce non-EU endpoints, notably in the United States and Asia. Still, we find that iOS applications strongly rely on connections to China and the United States. This difference implies that, although both platforms pose some privacy risk through non-EU data flows, the iOS ecosystem may be more exposed due to its significant engagement with countries outside the EU regulatory framework. In summary, while IoT devices seem to do well from a cross-border data transfer perspective, the companion apps, especially iOS ones, substantially increase the risk of transferring personal data to third countries without adequate data protection. Our findings highlight the importance of considering not only device behavior but also the role of companion apps in shaping the privacy landscape of IoT ecosystems.

7.7 Port Analysis

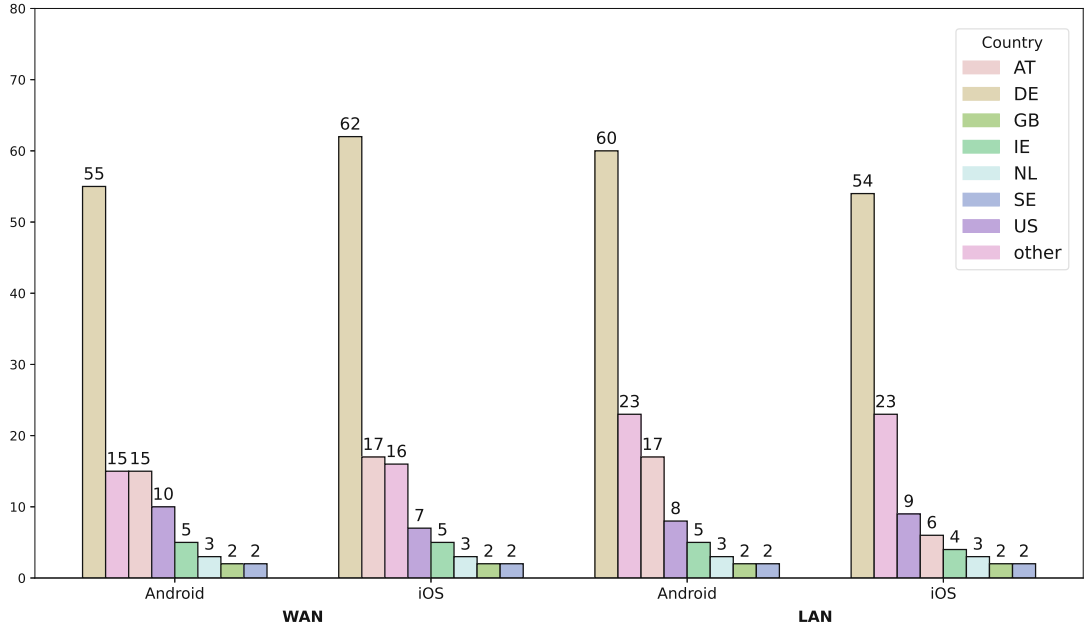
7.7.1 Android

The most frequently used port for Android across both companion apps and IoT devices is TCP 443 (HTTPS). We expect this as it supports secure communication over TLS/SSL and is commonly used by mobile apps and IoT devices to transmit telemetry or control data securely. The prevalence of port 443 indicates reliance on encrypted web traffic, which mitigates some risks of data interception. However, it does not eliminate concerns about transmitting data to third countries. The second most frequent port is UDP 3478, typically associated with the Session Traversal Utilities for NAT (STUN) protocol. In IoT contexts, STUN is often used to establish peer-to-peer (P2P) communication channels, particularly in smart home devices requiring real-time connectivity. In our dataset, this port is used primarily by the *Philips* baby monitor, which is consistent with its functionality, as baby monitors rely on low-latency audio and video streams that benefit from direct connections across NAT. TCP 8883, the default port for MQTT over TLS, also appears consistently. Its use with TLS provides confidentiality and integrity. Additionally, we observe port TCP 1443. While less standardized, certain vendors often use it as an alternative secure channel or proprietary service port. Its consistent use across devices and apps suggests vendor-specific implementation. Finally, UDP 123, the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.



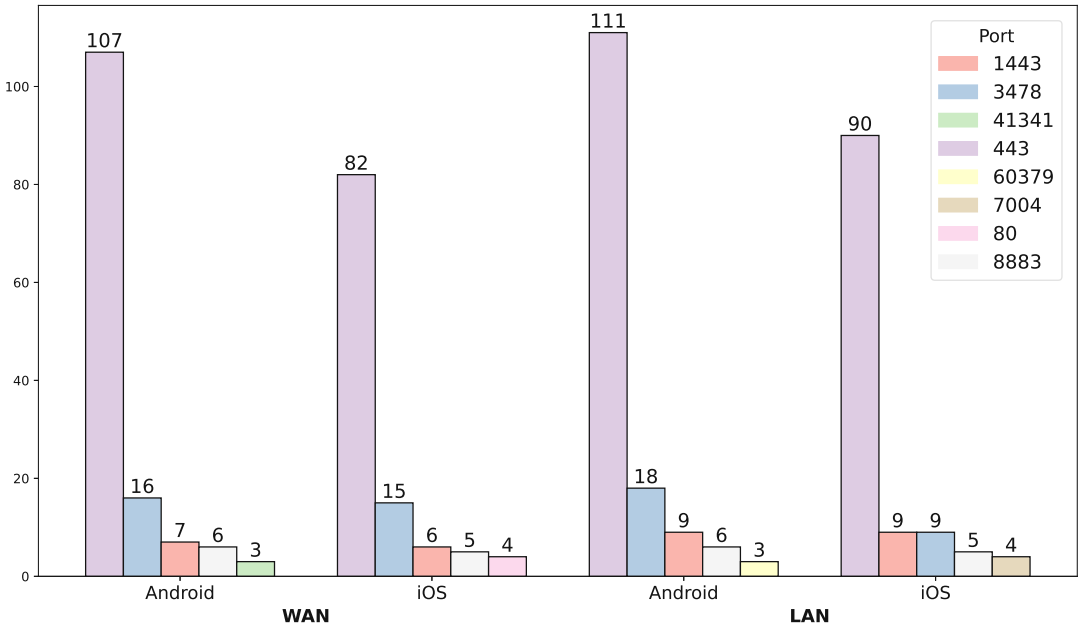
(a) Apps' countries



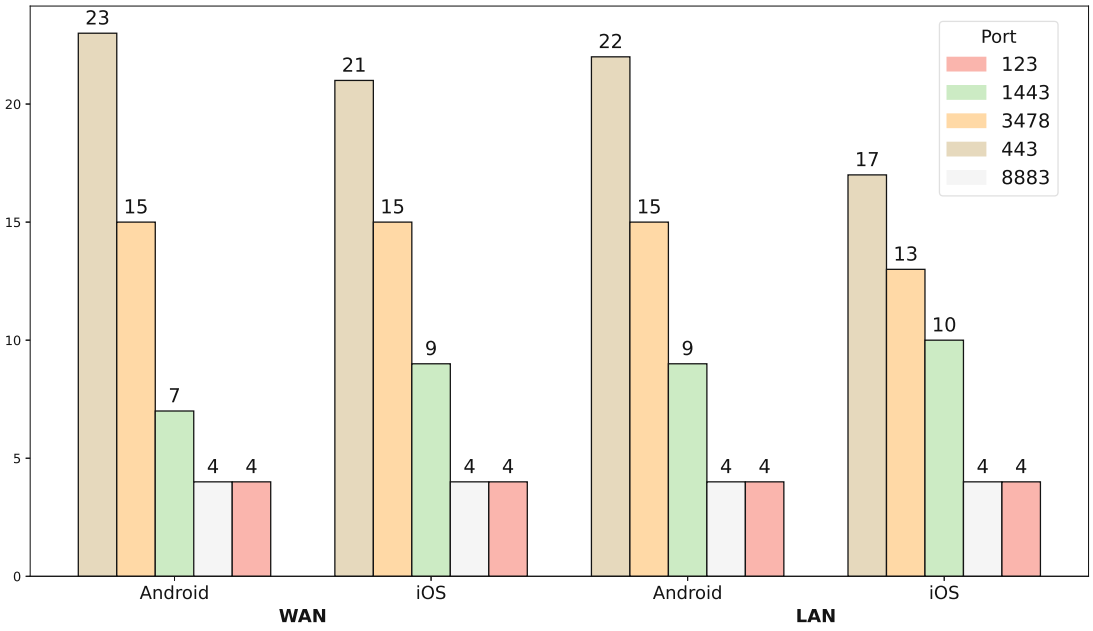
(b) Devices' countries

Figure 7.7: Top countries of endpoints in apps' and devices' traffic in the different OSs and network scenarios.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.



(a) Apps' top five ports.



(b) Devices' top five ports.

Figure 7.8: Top five adopted ports by endpoints in apps' and devices' traffic in the different OSs and network scenarios.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Network Time Protocol (NTP), is employed by Android devices. We also observe one occurrence of TCP 5222 in the *Xiaomi* app traffic, a port typically used by XMPP. While it only appears once and is therefore not shown in the plots, this finding illustrates that, besides MQTT, other IoT-relevant messaging protocols are also in use.

7.7.2 iOS

iOS shows a similar port usage pattern, though with some differences. Like Android, the most common port is TCP 443 (HTTPS). This reflects industry best practices. UDP 3478 (STUN) is the second most used port, reflecting iOS devices' reliance on NAT traversal for establishing real-time or peer-to-peer connections. TCP 8883 (MQTT over TLS) is also present, underscoring the protocol's cross-platform significance in IoT ecosystems. In contrast to Android, iOS devices show slightly higher usage of TCP 1443, again reflecting proprietary vendor-specific communication channels. The repeated presence of this port suggests a reliance on non-standardized secure transport. Finally, UDP 123 (NTP) is present in iOS device communication as well. This reinforces the importance of accurate time synchronization across IoT ecosystems. As in Android, we also find a single occurrence of TCP 5222 in the *Xiaomi* app traffic, again pointing to the use of XMPP.

7.7.3 Comparison

Both Android and iOS IoT ecosystems share dependence on ports TCP 443 (HTTPS), UDP 3478 (STUN), and TCP 8883 (MQTT over TLS). These ports represent pillars of secure IoT communication: encrypted web requests, real-time peer-to-peer signaling, and lightweight messaging. Their consistent presence across both platforms highlights common architectural choices in IoT ecosystems. Using standardized secure protocols (HTTPS, MQTT over TLS) is positive from a privacy and security standpoint. However, reliance on STUN servers and NTP infrastructure, particularly if operated outside the EU, introduces potential vectors for data exposure. We report our results in Figure 7.8.

7.8 MQTT Analysis

To gain deeper insights into the communication patterns of the IoT ecosystem, we extend the analysis to MQTT payloads. To this end, we first process the traffic recordings captured with *mitmproxy*. We then convert the raw mitmdump outputs into serialized Python objects using the pickle format, facilitating structured parsing of the captured traffic flows. We extract request–response bodies, headers, and parameters. This allows us not only the identification of functional message types (e.g., device telemetry, configuration updates, or keep-alive signals) but also an evaluation of whether sensitive data such as device identifiers, network attributes, or usage patterns are exposed in plaintext within these messages once decrypted. In addition, we inspect PCAP traces to examine communications over port 1883, the default port for unencrypted MQTT. Interestingly, we observe that the *Meross* device communicates via MQTT over port 443, repurposing

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

the standard HTTPS port instead of the usual MQTT defaults. We find that certain IoT devices transmit messages without encryption, making their payloads directly observable on the network. We show our results in Table 7.5.

We find that five apps adopt MQTT in their communication, i.e., *Cuco*, *Hombli*, *Philips*, *Teckin*, and *Xiaomi*. We parse their payloads from MQTT sessions, and systematically compare their structure (e.g., presence of JSON fields, readability, encoding) and content (e.g., device identifiers, state transitions). We also evaluate the persistence of identifiers across sessions and differences in data obfuscation between operating systems.

We reveal consistent JSON-structured payloads across Android devices in both scenarios. In the case of *Cuco*, *Hombli*, *Philips*, and *Teckin*, payloads share a common format containing fields such as clientId, deviceType, and userName. *Philips* exhibits significantly longer and more complex payloads compared to the other devices. *Xiaomi* directly encodes device state transitions into its payloads. For instance, ON and OFF commands are represented as structured JSON objects with keys, timestamps, and device identifiers, providing higher transparency. A further difference emerges when comparing network scenarios: while the first portion of the client identifier string changes between scenarios, the trailing portion remains the same. Unfortunately, all the information is either encoded in an unknown format or encrypted, thus making it impossible for us to understand the mutation patterns. In contrast, iOS devices produce largely opaque payloads. Across all tested vendors, most MQTT traffic appears as binary-encoded messages with little to no human-readable JSON content. Furthermore, the number and length of messages are significantly reduced compared to Android. *Xiaomi* devices produce no observable payloads in the iOS captures, contrasting with their Android counterpart.

Inspecting the payloads of the *Teckin* and *Levoit* devices did not yield any meaningful results as the messages they exchange are either malformed or appear to be ping requests to the MQTT broker.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

	Device Traffic				App Traffic			
	WAN		LAN		WAN		LAN	
	🤖	🍎	🤖	🍎	🤖	🍎	🤖	🍎
Teckin	🔓	🔓	🔓	🔓	🔒	🔒	🔒	🔒
Philips	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒
Xiaomi	✖	✖	✖	✖	🔒	✖	🔒	✖
Wiz	🔒	🔒	🔒	🔒	✖	✖	✖	✖
Hombli	✖	✖	✖	✖	🔒	🔒	🔒	🔒
CUCO	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒
Levoit	🔓	🔓	🔓	🔓	✖	✖	✖	✖
Meross(*)	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒

Table 7.5: In the table we show the eight devices (out of 10) that adopt the MQTT protocol. For each device or app, we only find one endpoint in each sub-scenario. We highlight with 🔓 the MQTT connections over the insecure port "1883" (in clear text) and with 🔒 the encrypted MQTT connections over port "8883". Further, when ✖ is found, we could not find any MQTT instance. We mark Meross with an asterisk "*" as it communicates with the MQTT protocol over the secure port "443" instead of the standard ones. We highlight how devices and apps exhibit different communication patterns, highlighting the complexity of the IoT ecosystem.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 8

PII and ID Sharing

Mobile apps are known for sharing PII of different natures, e.g., geolocation information, age, and gender, possibly but not limited to analytics and tracking purposes [7, 46, 69]. We deem it essential to thoroughly understand the relationship between IoT devices and companion apps to investigate which information is collected and shared. To this end, we inspect the payloads and API calls we find in the analyzable traffic we collect. Our results are based only on traffic obtained from companion apps. Although we collect traffic from the IoT devices as well, we could not bypass encryption since this would require altering each device's firmware.

8.1 PII Analysis

We first decrypt each capture from the *frida* run with the TLS session keys exported by *mitmproxy*. From the resulting PCAP files, we extract all HTTP and HTTP/2 URIs together with their payloads. Using the predefined PII of our fictitious persona (see Section 5.4) and device identifiers recorded during the experiments (e.g., IP and MAC addresses), we search the decrypted traffic for those exact strings and three common variations (lowercase, MD5-hashed, and Base64-encoded). We collect additional identifiers from the phones to broaden the set of device identifiers. We use the Android app *Device ID* (v1.3.0) [64] to retrieve all device identifiers associated with our Android phone. This app captures a range of IDs, including the Android ID, IMEI, Android Device ID, and the Advertising ID. On iOS, we use *My Device ID by AppsFlyer* [4] (v1.18.3), which returns the Identifier for Advertisers (IDFA). We copy the IMEI from Settings. We then search the decrypted traffic for all collected identifiers.

In this thesis, we define as PII (1) *Device Identifiers*, e.g., Mobile Equipment Identifiers (IMEIs), Advertiser ID, MAC addresses, SSIDs, (2) *User Identifiers*, which serve to identify a specific user, such as name, date of birth, email address, phone numbers, (3)

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Table 8.1: PII observed by platform and network setting. We report the number of apps and the share out of 10 IoT apps per platform. We see increased sharing of PII in the LAN scenario on both platforms. Cells marked with – indicate that we observed no instances. "Encrypted Data" denotes payloads unreadable after TLS decryption (custom app-layer encryption). "No PII found" denotes apps with no PII in the decrypted traffic.

	Android		iOS	
	LAN	WAN	LAN	WAN
MAC	3 (30%)	1 (10%)	2 (20%)	1 (10%)
SSID	–	–	–	–
IMEI	–	–	–	–
Email Address	1 (10%)	1 (10%)	1 (10%)	1 (10%)
Name	–	–	–	–
Geolocation	1 (10%)	–	–	–
IP	3 (30%)	1 (10%)	3 (30%)	2 (20%)
Username	–	–	–	–
Password	1 (10%)	1 (10%)	–	–
Date of Birth	–	–	–	–
Encrypted Data	5 (50%)	5 (50%)	5 (50%)	5 (50%)
No PII found	2 (20%)	4 (40%)	2 (20%)	3 (30%)

Location, e.g., geographical coordinates and IPs, and (4) *Credentials*, i.e., username and password.

We manually verify the extracted information to eliminate any false positives from our findings. We conduct our analysis on both network scenarios, LAN and WAN, for Android and iOS, to compare the impact of different network settings and mobile platforms on privacy. We summarize our results in Table 8.1. Even after decrypting TLS, we still find that half of the apps (five out of ten) apply custom encryption to their payloads. Therefore, we cannot analyze these apps for PII, and we label those cases as *Encrypted Data* in the table.

8.1.1 Android

We noticeably observe more PII on Android in the LAN scenario. For example, *Tapo* sends the geographical location to their first-party endpoints only in this scenario. Three of ten apps transmit the device's MAC address in the LAN scenario, compared to one in the WAN scenario. *Meross* also transmits the Wi-Fi access point MAC address (the BSSID) to its first-party endpoint. The BSSID is sensitive because it can map a BSSID to a location using crowd-sourced databases or commercial location services. For example, Google [21] documents an API that accepts nearby access point identifiers and returns

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

coordinates and an accuracy radius, and Apple [3] states that Location Services use crowd-sourced Wi-Fi hotspots to determine device location. Therefore, by knowing the BSSID, one can infer the router's physical location without GPS, which elevates the privacy risk. Similar to MAC addresses, we observe a similar drop for private IP addresses, which fall from three LAN occurrences to one on WAN. We also observe that *Tapo* transmits the user's email address and account password in both network scenarios.

8.1.2 iOS

iOS follows the same pattern, though overall PII transmission is lower. Two apps transmit the MAC address on the LAN, and only one does so on the WAN. Similar to MAC addresses, the occurrence of IP addresses decreases by one on the WAN compared to the LAN. One app shares an email address in both scenarios.

8.1.3 Comparison

Across both platforms, we observe more PII in the LAN scenario. The effect is stronger on Android, where PII instances rise from four on WAN to nine on LAN. On iOS, they rise from four to six. Besides network identifiers such as MAC and IP, which we observe on both platforms, we also see geolocation data and a password on Android.

8.2 ID Analysis

After manually inspecting reconstructed values in our previous analysis, we noticed a broad usage of non-standard identifiers, that is, not only Android or Advertiser IDs, which were shared across endpoints and companion apps. As our previous PII analysis does not encompass such non-standard values, we use regular expressions to extract all IDs from network requests and then compare them across all companion apps and endpoints to check whether they are shared or reused, potentially for advertising or tracking purposes. In fact, adopting the same set of IDs across different apps can pose significant risks to users' privacy, as aggregated data can be generated, leading to more precise user preferences. Such data is particularly interesting to analytics endpoints that can better tailor personalized content to the consumer. As for our previous analysis, we manually review the extracted IDs to remove any false positives.

We classify an ID match as interesting if it fulfills at least one of the following criteria: (1) the same ID is transmitted to more than one distinct endpoint, or (2) two or more companion apps share an ID. In certain cases, we expect to find the same IDs shared across different endpoints for one companion app, for example `account_id` appearing across TP-Link endpoints (`iac.tplinknbu.com` and `euw1-app-tapo-care.i.tplinknbu.com`). For this reason, we do not flag them as relevant. We summarize the result in Table 8.2.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

	Devices	ID Names	Endpoints
Android	Teckin Xiaomi	advertiser_id	graph.facebook.com
Android	Teckin	tid id	www.googletagmanager.com region1.analytics.google.com www.google.at
iOS	CUCO	traceId	ghome-smart-official.myshopify.com sentry.tuyaeu.com
iOS	CUCO	tid id	www.googletagmanager.com region1.google-analytics.com tpm.tuyaeu.com
iOS	CUCO	cid sid	tpm.tuyaeu.com region1.google-analytics.com

Table 8.2: ID Sharing Across Devices and Endpoints. We report the IDs reused by different devices and/or shared with different endpoints. We group the table by the same ID value, which can take different names as reported in column "ID Names". We report the app-device pairs in which such ID was found in the first column, while the latter shows which endpoints such ID was directed. We divide the table into two sections: (1) Android findings, (2) iOS findings.

8.2.1 Android

We identify one case where the same advertising identifier (`advertiser_id`) appears for devices from different manufacturers, namely *Teckin* and *Xiaomi*. *Teckin* also reuses the same identifier (`tid`/`id`) across three Google endpoints: `www.googletagmanager.com`, `region1.analytics.google.com`, and `www.google.at`. These entries show that Android devices share an advertising ID across manufacturers and send IDs to tracking and analytics services.

8.2.2 iOS

Unlike Android, we do not find cross-vendor sharing. All findings come from the *Cuco* app. *Cuco* sends several IDs to various analytics and support services. This keeps the sharing inside one app but shares those IDs with many parties.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

8.2.3 Comparison

Android links devices from different vendors through one advertising ID. iOS keeps reuse inside a single app but shares more types of IDs with more services. Both platforms send user IDs to analytic endpoints. The tracking risk exists in both cases, but Android increases linkability across brands while iOS increases exposure within the same app's services.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 9

Limitations & Future Work

9.1 Limitations & Future Work

Encrypted Device Traffic. One of the limitations we face is the inability to capture unencrypted traffic generated by IoT devices. In fact, this would require altering the firmware and possibly installing our proxy's custom Certificate Authority (CA) so that the device can trust it. Indeed, this procedure does not scale, introduces a high risk of bricking the device (thus making it unusable), or is impossible because of the lack of firmware images on the Internet. With only encrypted traffic, the analysis we can perform on the content of requests is limited only to payload size and frequency.

Limited PII Extraction. We currently detect only the PII we embed during user setup. However, we observed that many vendors (50% of our 10 tested devices) apply application-layer encryption to every payload, making PII analysis difficult. A prominent example is Tuya, which provides an IoT platform framework adopted by multiple vendors and accounts for 40% of our test devices. This reliance on a common backend complicates traffic analysis due to proprietary encryption and raises questions about ecosystem-level risks: vulnerabilities or misconfigurations in the Tuya infrastructure could simultaneously affect a large portion of the IoT market. Identifying and decoding Tuya payloads is therefore a crucial next step, and we plan to evaluate tools such as *tuyadump* [44] to support this task. We also think a more in-depth analysis of payloads, looking for API patterns and PII leakage, could unveil more differences or similarities with the apps' traffic. Therefore, we also plan to explore how machine learning and large language models can help extract PII from PCAP files.

Extend CapIoT. We plan to turn CapIoT into a plugin-based framework. Users can write custom runner modules describing their experiment flow using our capture pipeline and UI control mechanisms.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Extend Device Set. Our current device set covers a handful of product families. We plan to test more IoT devices from different categories, like smart vacuums, locks, and connected health products such as scales. This wider mix will let us test whether certain categories share more data. In particular, we want to learn whether devices that handle sensitive information, for example, health metrics, follow stricter PII protections.

Bluetooth Traffic. Many IoT devices pair Wi-Fi with Bluetooth Low Energy (BLE) and switch to BLE when they are nearby. We have already captured BLE packets, yet we have not parsed them. By analysing these logs, we can verify whether devices actually use BLE when the option is available.

IoT Lifecycle Coverage. Currently, we capture traffic only when we trigger a single function, e.g., turning a smart bulb on or off. However, a device may send other data using other features, so we plan to test every available function. For example, we toggle only the microphone in the camera experiments, yet the devices can also take snapshots and mute or unmute the speaker. We also plan to record and analyse traffic during each phase of the device lifecycle, which includes pairing, regular operation, factory reset, and account deletion, because each stage may expose different information.

Automated UI Interactions. Another limitation of the present study concerns the lack of automation in testing device and application functionalities. We have to perform interactions manually once, which restricts the scale and reproducibility of experiments. An automated framework capable of detecting the state of the user interface (e.g., recognizing what is displayed on the screen) and performing actions accordingly would allow us to explore device behavior more exhaustively. Such an approach would enable broader coverage of functionalities. Prior research in automated UI testing and dynamic analysis has suggested viable methodologies for such automation. For example, advanced UI-automation techniques, such as vision-language model–assisted systematic exploration, have been demonstrated to improve coverage of complex mobile application interfaces significantly [9]. Integrating these approaches in future work would substantially strengthen the robustness of the analysis.

9.2 Ethical Considerations

All our experiments are controlled, meaning there is no real user interaction, but we record and replay one functionality. While this might limit our exploration of apps' and devices' functionalities, thus preventing us from triggering all the traffic they can generate, potentially missing additional *Tracking & Analytics* endpoints, this ensures that no ethical boundaries are crossed. If users were involved, we would have to go through the Ethics Review Board (ERB) approval to ensure their safety and make them aware of the tests.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

CHAPTER 10

Conclusion

In this thesis, we investigate how IoT devices and their companion apps on Android and iOS communicate under different network scenarios, focusing on the privacy and security implications of these communication practices. Using our CapIoT framework, we analyze ten IoT devices and apps and analyze the resulting network traffic.

Our findings show that companion apps are the main culprits of communication complexity. While devices typically limit communication to vendor-controlled infrastructures, apps introduce numerous additional endpoints, including third-party tracking and analytics services.

We further show that local communication support is inconsistent. Although most devices establish direct LAN communication when possible, others, such as *Xiaomi*, *Levoit*, and *EchoDot v5*, rely exclusively on cloud endpoints even in LAN. This contradicts ENISA guidelines, which recommend preferring local communication to limit unnecessary data exposure.

In addition to endpoint counts, our analysis of cloud providers and endpoint geography shows heavy dependence on AWS across both ecosystems, but with significant differences in distribution: Android traffic leans strongly on Google services, while iOS apps exhibit stronger connections to vendor-operated infrastructures, including those located outside the EU (notably China and the US). These choices raise concerns about compliance with GDPR and data protection standards.

Our privacy analysis highlights additional risks. Both ecosystems transmit data to tracking and analytics endpoints, but in different ways: Android apps tend to show higher variability and, in one case (i.e., *Teckin*), a larger number of trackers per app, while iOS apps contact such endpoints more consistently across apps, with a strong concentration in the United States. We also observe widespread sharing of personally identifiable information (PII), often more prevalent in LAN than WAN, and the reuse of identifiers across devices and apps.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Our findings demonstrate that neither Android nor iOS offers consistently better privacy guarantees in the IoT ecosystem. Instead, the two platforms expose users to different but equally concerning risks. Based on our findings, we argue there is a need for improved security and privacy standards, greater transparency, and stricter regulatory oversight in developing and deploying IoT devices and their associated apps.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Overview of Generative AI Tools Used

I used Grammarly to improve the readability and grammar of sentences, and ChatGPT-5 to help with the creation of LaTeX tables.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Figures

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Tables

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Bibliography

[1] akabe1. *frida-multiple-unpinning*. Aug. 11, 2025. URL: https://codeshare.frida.re/@akabe1/frida-multiple-unpinning/ (visited on 08/11/2025).

[2] S. Andy, B. Rahardjo, and B. Hanindhito. „Attack scenarios and security analysis of MQTT communication protocol in IoT system“. In: *2017 4th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*. 2017. DOI: 10.1109/EECSI.2017.8239179.

[3] Apple Inc. *Location Services & Privacy*. URL: https://www.apple.com/legal/privacy/data/en/location-services/ (visited on 08/28/2025).

[4] AppsFlyer. *My Device ID by AppsFlyer*. 2025. URL: https://apps.apple.com/us/app/my-device-id-by-appsflyer/id1192323960 (visited on 08/29/2025).

[5] badmojr. *1Hosts*. May 12, 2024. URL: https://github.com/badmojr/1Hosts (visited on 08/25/2025).

[6] J. Chen, W. Diao, Q. Zhao, C. Zuo, Z. Lin, X. Wang, W. C. Lau, M. Sun, R. Yang, and K. Zhang. „IoTFuzzer: Discovering Memory Corruptions in IoT Through App-based Fuzzing“. In: *25th Network and Distributed System Security Symposium (NDSS)*. Feb. 2018. DOI: 10.14722/ndss.2018.23159.

[7] A. Continella, Y. Fratantonio, M. Lindorfer, A. Puccetti, A. Zand, C. Kruegel, and G. Vigna. „Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis“. In: *24th Network and Distributed System Security Symposium (NDSS)*. Feb. 2017.

[8] M. J. Davino, L. Melo, H. Lu, M. d'Amorim, and A. Prakash. „A Study of Vulnerability Analysis of Popular Smart Devices Through Their Companion Apps“. In: *IEEE Workshop on the Internet of Safe Things (SafeThings)*. May 23, 2019. DOI: 10.1109/SPW.2019.00042.

[9] B. F. Demissie, Y. N. Tun, L. K. Shar, and M. Ceccato. *VLM-Fuzz: Vision Language Model Assisted Recursive Depth-first Search Exploration for Effective UI Testing of Android Apps*. 2025. URL: https://arxiv.org/abs/2504.11675.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[10] A. Dilawer, D. Anupam, and Z. Fareed. „Analyzing the Feasibility and Generalizability of Fingerprinting Internet of Things Devices“. In: *22nd Privacy Enhancing Technologies Symposium (PETS)*. July 2022. DOI: 10.2478/popets-2022-0057.

[11] DomainTools. *Reverse IP Lookup*. 2025. URL: https://reverseip.domaintools.com/ (visited on 08/24/2025).

[12] D. Dragomir, L. Gheorghe, S. Costea, and A. Radovici. „A Survey on Secure Communication Protocols for IoT Systems“. In: *International Workshop on Secure Internet of Things (SIoT)*. Sept. 2016. DOI: 10.1109/SIoT.2016.012.

[13] Eclipse Foundation. *2024 IoT & Embedded Developer Survey Report*. 2024. URL: https://outreach.eclipse.foundation/iot-embedded-developer-survey-2024 (visited on 08/19/2025).

[14] P. Emami-Naeini, H. Dixon, Y. Agarwal, and L. F. Cranor. „Exploring How Privacy and Security Factor into IoT Device Purchase Behavior“. In: *ACM SIGCHI Conference on Human Factors in Computing Systems (CHI)*. May 2019. DOI: 10.1145/3290605.3300764.

[15] ENISA. *IoT Security Standards Gap Analysis*. Apr. 17, 2024. URL: https://www.enisa.europa.eu/publications/iot-security-standards-gap-analysis/@@download/fullReport (visited on 08/10/2025).

[16] Exodus. *Trackers*. Aug. 12, 2025. URL: https://reports.exodus-privacy.eu.org/en/trackers/ (visited on 08/12/2025).

[17] E. Faranda. *PCAPdroid*. Version 1.8.7. July 24, 2025. URL: https://github.com/emanuele-f/PCAPdroid (visited on 08/05/2025).

[18] O. Foundation. *WebDriverAgent*. URL: https://appium.github.io/appium-xcuitest-driver/4.16/wda-custom-server/ (visited on 08/23/2025).

[19] *Frida*. July 18, 2025. URL: https://frida.re/ (visited on 07/18/2025).

[20] Google. *Factory Images for Nexus and Pixel Devices*. July 8, 2025. URL: https://developers.google.com/android/images?hl=en#shiba (visited on 08/05/2025).

[21] Google Developers. *Geolocation API Overview*. Aug. 26, 2025. URL: https://developers.google.com/maps/documentation/geolocation/overview (visited on 08/28/2025).

[22] T. Group. *tcpdump & libpcap*. URL: https://www.tcpdump.org/ (visited on 08/31/2025).

[23] S. Gupta. *How to Root Android 14 Using Magisk?* Dec. 30, 2023. URL: https://www.getdroidtips.com/root-android-14-using-magisk/ (visited on 08/05/2025).

[24] hagezi. *DNS Blocklists*. Aug. 30, 2025. URL: https://github.com/hagezi/dns-blocklists (visited on 08/30/2025).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[25] S. Hollister. *Anker Finally Comes Clean about its Eufy Security Cameras*. Jan. 31, 2023. URL: https://www.theverge.com/23573362/anker-eufy-security-camera-answers-encryption (visited on 08/01/2025).

[26] Y. Jia, L. Xing, Y. Mao, D. Zhao, X. Wang, S. Zhao, and Y. Zhang. „Burglars' IoT Paradise: Understanding and Mitigating Security Risks of General Messaging Protocols on IoT Clouds". In: *41st IEEE Symposium on Security & Privacy (S&P)*. May 2020. DOI: 10.1109/SP40000.2020.00051.

[27] C. Jin, Y. Jia, Y. Song, Q. Tan, R. Yang, and Z. Liu. „Security and Privacy Measurement on Chinese Consumer IoT Traffic based on Device Lifecycle". In: (2025). DOI: 10.48550/arXiv.2505.09929.

[28] K. Kollnig, A. Shuba, R. Binns, M. Van Kleek, and N. Shadbolt. „Are iPhones Really Better for Privacy? A Comparative Study of iOS and Android Apps". In: *Proceedings on Privacy Enhancing Technologies* 2022 (Apr. 2022). DOI: 10.2478/popets-2022-0033.

[29] M. Komsic and C. Tagliaro. *CapIoT*. Sept. 2, 2025. URL: https://github.com/SecPriv/CapIoT (visited on 09/02/2025).

[30] F. Loi, A. Sivanathan, H. H. Gharakheili, A. Radford, and V. Sivaraman. „Systematically Evaluating Security and Privacy for Consumer IoT Devices". In: *Proceedings of the Workshop on Internet of Things Security and Privacy (IoTS&P)*. 2017. DOI: 10.1145/3139937.3139938.

[31] 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: *21st Privacy Enhancing Technologies Symposium (PETS)*. July 2021. DOI: 10.2478/popets-2021-0075.

[32] E. Al-Masri, K. R. Kalyanam, J. Batts, J. Kim, S. Singh, T. Vo, and C. Yan. „Investigating Messaging Protocols for the Internet of Things (IoT)". In: *IEEE Access* 8 (2020). DOI: 10.1109/ACCESS.2020.2993363.

[33] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A.-R. Sadeghi, and S. Tarkoma. „IoT SENTINEL: Automated Device-Type Identification for Security Enforcement in IoT". In: *37th IEEE International Conference on Distributed Computing Systems (ICDCS)*. June 2017. DOI: 10.1109/ICDCS.2017.283.

[34] *mitmproxy*. Aug. 4, 2023. URL: https://mitmproxy.org/ (visited on 04/14/2024).

[35] mitmproxy. *How mitmproxy works*. Apr. 25, 2025. URL: https://docs.mitmproxy.org/stable/concepts/how-mitmproxy-works/ (visited on 08/19/2025).

[36] NIST. *Cybersecurity for IoT Program*. Accessed: October 8, 2023. 2021. URL: https://www.nist.gov/itl/applied-cybersecurity/nist-cybersecurity-iot-program/consumer-iot-cybersecurity.

[37] NVISO. *Always Trust User Certs*. May 21, 2025. URL: https://github.com/NVISOsecurity/AlwaysTrustUserCerts (visited on 08/05/2025).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[38] OASIS. *MQTT Version 3.1.1.* Tech. rep. Version 3.1.1. Last accessed 16-08-2025. OASIS, 2014. URL: https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html.

[39] T. OConnor, R. Mohamed, M. Miettinen, W. Enck, B. Reaves, and A.-R. Sadeghi. „HomeSnitch: Behavior Transparency and Control for Smart Home IoT Devices". In: *12th ACM Conference on Security & Privacy in Wireless and Mobile Networks (WISEC)*. May 2019. DOI: 10.1145/3317549.3323409.

[40] openatx. *facebook-wda*. URL: https://github.com/openatx/facebook-wda (visited on 08/30/2025).

[41] *palera1n*. Version: v2.0.2. URL: https://palera.in/ (visited on 08/23/2025).

[42] R. Perdisci, T. Papastergiou, O. Alrawi, and M. Antonakakis. „IoTFinder: Efficient Large-Scale Identification of IoT Devices via Passive DNS Traffic Analysis". In: *5th IEEE European Symposium on Security & Privacy (EuroS&P)*. Sept. 2020. DOI: 10.1109/EuroSP48549.2020.00037.

[43] A. Pradeep, M. T. Paracha, P. Bhowmick, A. Davanian, A. Razaghpanah, T. Chung, M. Lindorfer, N. Vallina-Rodriguez, D. Levin, and D. Choffnes. „A Comparative Analysis of Certificate Pinning in Android & iOS". In: *22nd Internet Measurement Conference (IMC)*. Oct. 2022. DOI: 10.1145/3517745.3561439.

[44] py60800. *tuyadump*. URL: https://github.com/py60800/tuyadump (visited on 08/31/2025).

[45] S. Ramnath, A. Javali, B. Narang, P. Mishra, and S. K. Routray. „IoT-based Localization and Tracking". In: *Proceedings of the International Conference on IoT and Application (ICIOT)*. 2017. DOI: 10.1109/ICIOTA.2017.8073629.

[46] 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". In: *25th Network and Distributed System Security Symposium (NDSS)*. Feb. 2018. DOI: 10.14722/ndss.2018.23009.

[47] N. Redini, A. Continella, D. Das, G. De Pasquale, N. Spahn, A. Machiry, A. Bianchi, C. Kruegel, and G. Vigna. „DIANE: Identifying Fuzzing Triggers in Apps to Generate Under-constrained Inputs for IoT Devices". In: *40th IEEE Symposium on Security & Privacy (S&P)*. May 2019. DOI: 10.1109/SP40001.2021.00066.

[48] T. Reed. *New iOS exploit checkm8 allows permanent compromise of iPhones*. Sept. 27, 2019. URL: https://www.malwarebytes.com/blog/news/2019/09/new-ios-exploit-checkm8-allows-permanent-compromise-of-iphones (visited on 08/23/2025).

[49] J. Ren, D. J. Dubois, D. Choffnes, A. M. Mandalari, R. Kolcun, and H. Haddadi. „Information Exposure for Consumer IoT Devices: A Multidimensional, Network-Informed Measurement Approach". In: *19th Internet Measurement Conference (IMC)*. Oct. 2019. DOI: 10.1145/3355369.3355577.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[50] J. Ren, M. Lindorfer, D. J. Dubois, A. Rao, D. Choffnes, and N. Vallina-Rodriguez. „Bug Fixes, Improvements, ... and Privacy Leaks - A Longitudinal Study of PII Leaks Across Android App Versions“. In: *25th Network and Distributed System Security Symposium (NDSS)*. Feb. 2018. DOI: 10.14722/ndss.2018.23143.

[51] P. Saint-Andre. *Extensible Messaging and Presence Protocol (XMPP): Core*. RFC 6120. Mar. 2011. URL: https://www.rfc-editor.org/info/rfc6120.

[52] D. Schmidt, C. Tagliaro, K. Borgolte, and M. Lindorfer. „IoTFlow: Inferring IoT Device Behavior at Scale through Static Mobile Companion App Analysis“. In: *30th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. Nov. 2023. DOI: 10.1145/3576915.3623211.

[53] Sensepost. *objection - Runtime Mobile Exploration*. Apr. 6, 2021. URL: https://github.com/sensepost/objection (visited on 08/12/2025).

[54] A. Sharif. *Consumer Spending on Smart Home Products and Services Worldwide from 2015 to 2025*. July 1, 2025. URL: https://www.statista.com/statistics/693303/smart-home-consumer-spending-worldwide/ (visited on 08/01/2025).

[55] A. Sivanathan, H. H. Gharakheili, F. Loi, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman. „Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics“. In: *IEEE Transactions on Mobile Computing* 18.8 (2019). DOI: 10.1109/TMC.2018.2866249.

[56] M. Steinböck, J. Troost, W. van Beijnum, J. Seredynski, H. Bos, M. Lindorfer, and A. Continella. „SoK: Hardening Techniques in the Mobile Ecosystem—Are We There Yet?“ In: *10th IEEE European Symposium on Security & Privacy (EuroS&P)*. June 2025.

[57] C. Tagliaro, M. Komsic, A. Continella, K. Borgolte, and M. Lindorfer. „Large-Scale Security Analysis of Real-World Backend Deployments Speaking IoT-Focused Protocols“. In: *27rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*. Oct. 2024. DOI: 10.1145/3678890.3678899.

[58] R. Trimananda, J. Varmarken, A. Markopoulou, and B. Demsky. „Packet-Level Signatures for Smart Home Devices“. In: *27th Network and Distributed System Security Symposium (NDSS)*. Feb. 2020. DOI: 10.14722/ndss.2020.24097.

[59] N. Vallina-Rodriguez, J. Shah, A. Finamore, Y. Grunenberger, K. Papagiannaki, H. Haddadi, and J. Crowcroft. „Breaking for Commercials: Characterizing Mobile Advertising“. In: *12th Internet Measurement Conference (IMC)*. Nov. 2012. DOI: 10.1145/2398776.2398812.

[60] J. Varmarken, H. Le, A. Shuba, Z. Shafiq, and A. Markopoulou. „The TV is Smart and Full of Trackers: Measuring Smart TV Advertising and Tracking“. In: *20th Privacy Enhancing Technologies Symposium (PETS)*. July 2020. DOI: 10.2478/popets-2020-0021.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[61] ViRb3. *MagiskFrida*. July 8, 2025. URL: https://github.com/ViRb3/magisk-frida (visited on 08/05/2025).

[62] H. Wang, D. Xiong, P. Wang, and Y. Liu. „A Lightweight XMPP Publish/Subscribe Scheme for Resource-Constrained IoT Devices“. In: *IEEE Access* 5 (2017). DOI: 10.1109/ACCESS.2017.2742020.

[63] X. Wang, Y. Sun, S. Nanda, and X. Wang. „Looking from the Mirror: Evaluating IoT Device Security through Mobile Companion Apps“. In: *28th USENIX Security Symposium (USENIX Security)*. Aug. 2019. DOI: 10.5555/3361338.3361418.

[64] Wenxiang Zhang. *Device ID*. 2025. URL: https://play.google.com/store/apps/details?id=tw.reh.deviceid (visited on 08/29/2025).

[65] J. Wu. *Magisk*. May 14, 2025. URL: https://github.com/topjohnwu/Magisk (visited on 08/05/2025).

[66] M. Zhang. *Top 10 Cloud Service Providers Globally in 2024*. Apr. 18, 2024. URL: https://dgtlinfra.com/top-cloud-service-providers/ (visited on 08/02/2025).

[67] S. Zheng, N. Apthorpe, M. Chetty, and N. Feamster. „User Perceptions of Smart Home IoT Privacy“. In: *ACM Human-Computer Interaction* 2.CSCW (Nov. 2018). DOI: 10.1145/3274469.

[68] W. Zhou, Y. Jia, Y. Yao, L. Zhu, L. Guan, Y. Mao, P. Liu, and Y. Zhang. „Discovering and Understanding the Security Hazards in the Interactions between IoT Devices, Mobile Apps, and Clouds on Smart Home Platforms“. In: *28th USENIX Security Symposium (USENIX Security)*. Aug. 2019. DOI: 10.5555/3361338.3361417.

[69] C. Zuo, Z. Lin, and Y. Zhang. „Why Does Your Data Leak? Uncovering the Data Leakage in Cloud from Mobile Apps“. In: *40th IEEE Symposium on Security & Privacy (S&P)*. May 2019. DOI: 10.1109/SP.2019.00009.

[70] E. Zveare. *Hacking into the Worldwide Jacuzzi SmartTub Network*. June 20, 2022. URL: https://eaton-works.com/2022/06/20/hacking-into-the-worldwide-jacuzzi-smarttub-network/ (visited on 08/02/2025).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN