



IoTFlow: Inferring IoT Device Behavior at Scale through Static Mobile Companion App Analysis

David Schmidt
TU Wien
dschmidt@seclab.wien

Carlotta Tagliaro
TU Wien
carlotta@seclab.wien

Kevin Borgolte
Ruhr University Bochum
kevin.borgolte@rub.de

Martina Lindorfer
TU Wien
martina@seclab.wien

ABSTRACT

The number of “smart” devices, that is, devices making up the Internet of Things (IoT), is steadily growing. They suffer from vulnerabilities just as other software and hardware. Automated analysis techniques can detect and address weaknesses before attackers can misuse them. Applying existing techniques or developing new approaches that are sufficiently general is challenging though. Contrary to other platforms, the IoT ecosystem features various software and hardware architectures.

We introduce IoTFlow, a new static analysis approach for IoT devices that leverages their mobile companion apps to address the diversity and scalability challenges. IoTFlow combines Value Set Analysis (VSA) with more general data-flow analysis to automatically reconstruct and derive how companion apps communicate with IoT devices and remote cloud-based backends, what data they receive or send, and with whom they share it. To foster future work and reproducibility, our IoTFlow implementation is open source.

We analyze 9,889 manually verified companion apps with IoTFlow to understand and characterize the current state of security and privacy in the IoT ecosystem, which also demonstrates the utility of IoTFlow. We compare how these IoT apps differ from 947 popular general-purpose apps in their local network communication, the protocols they use, and who they communicate with. Moreover, we investigate how the results of IoTFlow compare to dynamic analysis, with manual and automated interaction, of 13 IoT devices when paired and used with their companion apps. Overall, utilizing IoTFlow, we discover various IoT security and privacy issues, such as abandoned domains, hard-coded credentials, expired certificates, and sensitive personal information being shared.

CCS CONCEPTS

• Security and privacy → Systems security.

KEYWORDS

Internet of Things (IoT); IoT security; IoT privacy; companion apps; network analysis; static analysis.

ACM Reference Format:

David Schmidt, Carlotta Tagliaro, Kevin Borgolte, and Martina Lindorfer. 2023. IoTFlow: Inferring IoT Device Behavior at Scale through Static Mobile Companion App Analysis. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23)*, November 26–30, 2023, Copenhagen, Denmark. ACM, New York, NY, USA, 15 pages. <https://doi.org/10.1145/3576915.3623211>



This work is licensed under a Creative Commons Attribution 4.0 International License. CCS '23, November 26–30, 2023, Copenhagen, Denmark.
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0050-7/23/11.
<https://doi.org/10.1145/3576915.3623211>

1 INTRODUCTION

The number of Internet of Things (IoT) devices, that is, smart devices, is rising rapidly: Forecasts expect the number of IoT devices to grow to 25.4 billion in 2030 [45]. These devices collect data about their users and environment to make smart decisions. For example, to call for help in an emergency, a smartwatch may collect health indicators. This means that users need to trust them to handle their data with care. Unfortunately, smart devices have gained notoriety for their security and privacy issues, leading to the catchphrase “*the S in IoT stands for security.*” Notably, employees of Ring had unauthorized access to users’ security camera footages uploaded to their cloud backend [54]. Similarly, the European Union (EU) recalled kids’ smartwatches because they exposed sensitive information and could be easily compromised by attackers [16].

Prior work extensively analyzed open and closed source desktop and mobile applications (apps) for security and privacy issues, but analyzing smart devices remains an open challenge. Related work in this domain mainly focused on firmware vulnerabilities [21, 27] or on analyzing a handful of selected devices [23, 44, 60, 81, 91, 96]. This does, however, not scale to the wide variety of smart devices with diverse software and hardware architectures. Intuitively, buying thousands of devices to analyze them in a lab setting is financially and practically infeasible.

Therefore, to enable the large-scale discovery and analysis of security and privacy issues in the IoT ecosystem, we propose IoTFlow, a novel static analysis approach for IoT devices via their mobile companion apps. These apps play an important role in controlling IoT devices directly and can serve as intermediaries to their cloud backends. Practically all IoT devices have such apps available for Android and iOS [21, 62, 63, 80]. They allow users to setup and control their devices locally, via the local network or Bluetooth, or remotely, via the Internet. For some devices, their apps are the only gateway to the Internet. Overall, the apps store and process information collected by the IoT devices and about the remote infrastructure. Given the nature of data that the devices collect and use, it may also be highly sensitive. Further, attackers could misuse apps with hard-coded information (e.g., endpoints, credentials) to eavesdrop on others’ private information, or distribute malicious content via misconfigured IoT backends or abandoned domains. Using a misconfigured backend, they could exploit vulnerabilities to create a new botnet of hundreds of thousands of devices, even if the devices are not directly reachable on the Internet.

The basic idea of evaluating the security and privacy of IoT devices indirectly by studying their companion apps has been explored by prior work. For example, Wang et al. [94] leveraged it to identify rebranded devices by searching for similar apps. They find vulnerabilities in other devices because of “private labeling” and component re-use. Chen et al. [21] and Redini et al. [80] used companion apps to inform fuzzing IoT devices, while Zuo et al. [102],

Sivakumaran et al. [85, 86], and Zhao et al. [99] leveraged companion apps to identify Bluetooth Low Energy (BLE) issues. Wang et al. [93] statically analyzed Samsung SmartThings apps, which are part of the SmartThings smart hub IoT ecosystem.

Existing approaches focus on re-identifying already known issues shared among multiple devices (previously discovered through traditional techniques), still require physical devices (fuzzing), focus on a subset of companion apps (BLE), or analyze conceptually simple apps that are less widespread than general companion apps [64] (e.g., Samsung SmartThings apps, which are event flow graphs, rather than full apps; similar to “If This Then That” [47]).

In this paper, we introduce a new static analysis approach, IoTFlow, that substantially advances this basic idea. Our new approach enables us to gain new fundamental knowledge about the companion apps and corresponding smart devices at scale without actually requiring the physical device. We focus on addressing two crucial limitations of state-of-the-art techniques: First, we discover new issues automatically instead of re-identifying existing issues, which would require a priori knowledge that they exist. Second, we investigate individual devices instead of assuming that groups of devices share or re-use components. Specifically, our approach enables us to infer and gain new insights into the security and privacy of companion apps and their corresponding smart devices by reconstructing information about the used network protocols, endpoints, and the data they receive. With our approach, we can answer the following important but open questions concerning security and privacy in the IoT ecosystem:

RQ1: *How do companion apps and devices communicate?*

RQ2: *Who are companion apps communicating with?*

RQ3: *Which data are companion apps sharing (and how)?*

Specifically, our approach (1) identifies communication trigger points, (2) uses Value Set Analysis (VSA) to reconstruct network-related information on where data is coming from or transferred to, such as the URLs that are being contacted, (3) utilizes Data-flow Analysis (DFA) to determine what data is being accessed, shared, and with whom, and (4) assesses the corresponding impact.

We evaluate our approach on 9,889 unique and manually verified companion apps [50, 62, 63] to show that we can analyze IoT devices accurately and at scale. Additionally, we study the differences in network behavior between the companion apps and 947 popular general-purpose apps that we collected. Finally, we verify the accuracy of IoTFlow and compare it with dynamic analysis, for which we interacted with 13 IoT devices via their companion apps.

In this paper, we make the following contributions:

- We introduce IoTFlow, a new static program analysis approach utilizing Value Set Analysis (VSA) and Data-flow Analysis (DFA) to analyze the behavior of IoT devices based on their companion apps’ interactions with them and their remote backend.
- We show that IoTFlow can accurately infer the network behavior of companion apps at scale by analyzing 9,889 IoT apps.
- We analyze how and with whom companion apps communicate, what data they share locally with devices and remotely, and we highlight their differences to general apps.
- Using IoTFlow, we automatically discover rampant security and privacy issues in the IoT ecosystem, such as abandoned control domains, hard-coded credentials, expired certificates, or shared Personally Identifiable Information (PII).

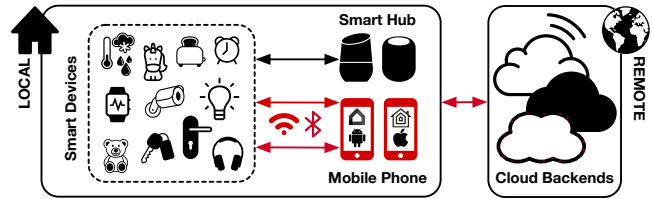


Figure 1: Overview of the IoT ecosystem and its command and control scenarios, including apps as intermediaries.

Artifacts. To foster reproducibility and future research, we make our open source implementation and analysis artifacts available at <https://github.com/SecPriv/iotflow>.

2 MOTIVATION

Following, we motivate IoTFlow with the need for at-scale IoT device behavior analysis, the interdependence of companion apps and IoT infrastructure, and the unique features of companion apps compared to general-purpose apps.

Large-scale IoT Device Behavior Analysis. The plethora of security and privacy issues that supposedly plague smart devices are a well-hypothesized problem in the security community and often anecdotally confirmed when yet another real-world issue is found and the press is reporting on it. Unfortunately, we currently lack techniques to discover such issues and also other vulnerabilities in smart devices automatically and at scale. State-of-the-art approaches focus on analyzing the devices’ firmware [27], requiring tedious and substantial manual effort to tailor it to each individual device, possibly even each hardware revision of a device. It also suffers from the many challenges of analyzing firmware, such as having to deduce and infer what sensors and actuators exist, model them, and understand how the firmware is communicating with it. Even if it would be feasible to scale such approaches to the many devices, it is also challenging to automatically gather thousands of firmware images, as devices use different processes to retrieve and update their firmware. At the same time, more and more IoT devices are being manufactured and used. Thus, it remains an open problem how to analyze the increasing number of diverse devices.

For large open source projects, the average lifetime of vulnerabilities is multiple years [1, 55]. Considering the profit-driven nature of the IoT ecosystem, it appears likely that security is indeed an afterthought in the IoT ecosystem and vulnerabilities might remain unpatched similarly long or even longer. Automated large-scale analysis allows us to promptly identify vulnerabilities and mitigate them. Moreover, even when automated analysis cannot replace in-depth analysis, it still helps developers to identify issues and address them. Being able to accurately analyze how IoT devices truly behave also informs privacy policy and behavior of (privacy-conscious) consumers. Practical large-scale automated analysis provides the much-needed foundation and knowledge to better understand IoT devices and improve their security and privacy.

IoT Control Infrastructure. The fundamental idea of smart devices is that they coordinate and cooperate with other devices, that is, they do not work in isolation. Typically, the devices communicate with companion apps, smart hubs, or remote cloud-based backends (see Figure 1), the latter of which may distributed over different

regions world-wide [83]. Users interact with the devices almost exclusively via their companion apps. If a device supports Wi-Fi, then the app may communicate with the device over the local network or the Internet. If a device does not support Wi-Fi but only uses Bluetooth, then all device-to-cloud communication needs to pass through the app or a hub. Moreover, due to missing user interfaces, updating a device’s firmware frequently happens via the app [94]. That is, the apps play a central role during the *setup*, *operation*, and *update* of the devices. In fact, many devices cannot be set up without using a device that can run the app. Thus, apps must contain some information about the devices and their behavior, and they provide a unique analysis opportunity.

General-purpose Apps vs. Companion Apps. Compared to general-purpose apps, companion apps face different challenges and introduce new threats. Generally, mobile operating systems restrict access to sensitive data and sensors (e.g., through Android or iOS permissions). However, this does not apply to data collected through smart devices. Users also lack visibility and control over the data the devices collect and share. It is crucial to investigate the threat of collusion between device and app, especially because it circumvents existing defenses and allows to build more accurate user profiles by combining PII and data collected by both [82].

Advertisements (ads) and trackers to collect user data for behavioral targeting appear widely in general-purpose apps [79, 82, 90]. These services are attractive for developers to monetize their apps [41]. For companion apps, one might assume that the business model centers around selling the devices. However, related studies showed that these apps and even devices themselves include ads and tracking [60, 81, 91]. In hindsight, considering the IoT environment and collusion potential, this makes sense: It is additional income. For example, companion apps can interact with the local network to discover and manage devices (a permission often required to set up the device), which is data general-purpose apps have difficulty to collect, and which is also useful for advertisement or tracking [52]. Prior work on network behavior and PII leakage of apps mainly considers traffic sent to remote servers. For IoT devices that use local communication, via Bluetooth or Wi-Fi, app-to-device or device-to-app communication has additional significance [87]. A smart device only using Bluetooth can collude with a companion app to “clean” sensitive data: receive it, encode it in some way, and send it back to the app, which sends it to the tracker. Existing ways to identify and block such behavior in general-purpose apps cannot address the challenges of the IoT environment, like collusion.

3 IOTFLOW

We introduce IoTFlow, a new static analysis approach for companion apps. We aim to better understand the behavior of IoT devices without requiring the physical device.

IoTFlow itself has two main phases (see Figure 2): Value Set Analysis (VSA) and Data-flow Analysis (DFA). With VSA, we identify *trigger points*, that is, sources and sinks of interesting (network) activities. This appears trivial at first, but it is important to realize that (1) we expect a substantial amount of communication, as smart devices are meant to communicate and coordinate extensively, and (2) we need to be able to determine the communication endpoint. For example, a user might expect and accept that the companion

app shares their location to turn on their heating when they are on their way home. But, most users would likely object if it is sent to an advertisement company. Enumerating all potential sources and sinks will lead to inaccurate results and render the analysis impractical. Instead, we need to distinguish where apps send data, to the device or a remote service, to which services, and utilizing which network protocols. We accurately reconstruct this information leveraging VSA (Section 3.1) and use it to identify precise sources and sinks for our DFA (Section 3.2).

For reconstructed endpoints, which may be third-party services, we then (1) categorize them based on their purpose, (2) analyze their geographic locations, and (3) test for abandoned domains. This allows us to evaluate if communication would be expected and assess their security and privacy impact. For example, a privacy-conscious user within the EU may not expect that their device sends data to a country not bound to the GDPR. Similarly, abandoned domains can lead to devices being taken over by attackers [17, 24, 74].

With DFA, we can then precisely assess which data companion apps share, with whom they communicate, and how. Specifically, we analyze the data-flow for data from the identified and categorized trigger points as well as from sensitive data sources (e.g., GPS location) to relevant sinks.

Motivating Example. Considering the examples in Listing 1 and Listing 2, we (1) need to reconstruct the destination of the Message Queuing Telemetry Transport (MQTT) broker (Listing 2, line 15), and (2) trace the data flow from the Bluetooth source (Listing 1, line 3) to where the message is published (Listing 2, line 18). An additional challenge is that the data is passed from Listing 1 line 6 to Listing 2 line 7 via Inter Component Communication (ICC). Traditional approaches would miss this example. However, we can reconstruct the keys of the ICC during VSA and then bridge the connection via the reconstructed keys, enabling us to perform more precise DFA across the ICC boundary.

3.1 Value Set Analysis

Value Set Analysis (VSA) is a program analysis technique to reconstruct values at specific program points. We utilize it to gain insights about the communication of IoT apps and to accurately handle ICC for our DFA. VSA has been used by related work before [101, 102], however, with the focus on reconstructing Application Programming Interface (API) keys or Universally Unique Identifiers (UUIDs) of BLE to identify vulnerable implementations of the BLE pairing process. That is, related work reconstructed primarily strings using manually derived rules, while IoTFlow supports arbitrary objects (as is required to precisely reconstruct endpoints, like in Listing 2).

Pre-Processing **1**. We implemented our IoTFlow prototype in Java and target Android. We use Soot [89] to parse Dalvik byte code from Android apps. It translates the byte code into the Jimple Intermediate Representation (IR), which simplifies our analysis (e.g., by splitting nested instructions). Notably, both Kotlin and Java Android apps are compiled into Dalvik code, and, in turn, IoTFlow can readily analyze both types of apps. In preparation for the forward computation step, we also translate the Dalvik byte code into Java byte code with dex2jar [71] because Java cannot load classes directly from the Dalvik byte code.

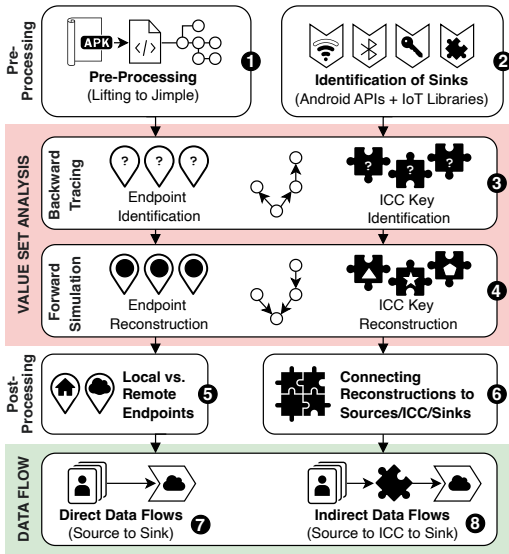


Figure 2: Overview of IoTFlow. We use VSA to reconstruct endpoints, cryptographic data, and ICC keys for the flow analysis. We use flow analysis to find data leaks, and connect request/response data with endpoints. With the ICC information of the VSA, we support data flows involving ICC.

Identification of Sinks ②. IoTFlow starts at interesting sinks tracing backward their values. We analyze network-related sinks from Android, Java, and 19 manually selected popular network communication libraries, focusing on IoT application layer messaging protocols (e.g., MQTT, Constrained Application Protocol (CoAP), Advanced Message Queuing Protocol (AMQP), and Extensible Messaging and Presence Protocol (XMPP)) [11, 70]. Additionally, we consider ICC and cryptographic methods as sinks. We later use the reconstructed ICC information to bridge the ICC boundary during DFA. As apps might encrypt data before sending it, we also examine cryptographic methods.

Backward Tracing ③. We then trace back through the program, starting at the identified sinks to all program points where the app modifies the values we are interested in. Naturally, this yields an over-approximate trace set. For example, if we want to reconstruct the parameter passed to `MqttManager` in Listing 2, then our reconstruction starts at line 15 (following $\leftarrow \rightarrow$). We trace back the value of `config.endpoint` to line 14, to line 8, to lines 1–3, until we have traced all variables on which `config` depends.

Forward Simulation ④. In the next step, we reconstruct the actual value set. Here, we must reconstruct arbitrary objects passed to the sinks, or we would miss the value of `config` in our example. That is, only reconstructing string operations is insufficient. Instead, we adapt our value reconstruction to handle arbitrary objects from any classes defined by the app, such as the `MqttConfig` class. We utilize reflection and forward simulate the backward trace, using the classes and methods as the app would do while normally executing it. Using reflection for simulating execution paths has a further advantage: We can handle code where the app itself uses reflection, which prior work cannot. However, reflection also introduces new challenges that we need to address:

```

1  →String BLE_DATA = "device";
2  @Override
3  void onCharacteristicRead(BluetoothGattCharacteristic bgc, /*...*/) {
4      Intent intent = new Intent(DeviceActivity.class)
5      byte[] value = bgc.getValue(); ←
6      intent.putExtra(BLE_DATA, parseData(value)); ←
7      this.startActivity(intent);
8  }
    
```

Listing 1: Simplified example code that reads device data via BLE and sends it via an Intent (ICC). Arrows on the left show VSA, and arrows on the right show DFA. We reconstruct the ICC key “device,” marked yellow, via VSA $\leftarrow \rightarrow$ (line 1). The \rightarrow arrows show the data flow from source to the ICC sink, via purple statements. The flow continues green in Listing 2.

```

1  MqttConfig config = new MqttConfig();
2  config.setEndpoint("example.com");
3  config.setTopic("things/Wifi_device");
4  class DeviceActivity {
5      @Override
6      void onCreate(Bundle bundle) {
7          String data = getIntent().getStringExtra(BLE_DATA);
8          Mqtt mqtt = new Mqtt(config);
9          mqtt.publish(new MqttMessage(data, config.topic); ←
10     }
11     }
12     class Mqtt {
13         MqttManager mqttManager;
14         Mqtt(MqttConfig config) {
15             this.mqttManager = new MqttManager(config.endpoint); ←
16         }
17         void publish(MqttMessage m) { ←
18             this.mqttManager.publishString(m.data, m.topic); ←
19         }
20     }
    
```

Listing 2: Simplified example code of an activity that receives BLE data and publishes it via MQTT. Arrows on the left show identification and reconstruction via VSA, marked $\leftarrow \rightarrow$. Arrows on the right show DFA. Connecting reconstructions are marked \rightarrow , via blue statements. The data flow from ICC source to sink, completing the flow from source to ICC sink of Listing 1, is marked \rightarrow , through green statements. We highlight the reconstructed ICC key “device” yellow again.

- (1) Android Methods.** Some data might not be available statically, like user input. Additionally, we cannot simulate Android methods with reflection because only stub implementations are available and we use placeholders instead (e.g., intents, shared preferences, and database).
- (2) Non-Terminating Methods.** Simulating arbitrary methods with reflection can also lead to non-termination, such as when it waits for an IoT device to connect. We mitigate this issue by terminating it after two seconds. We determined this threshold empirically as a trade-off between precision and time. In practice, most instructions finish within a fraction of a second.
- (3) Partially Reconstructed Values.** Partially reconstructed values can cause us to miss values. We may simulate a substring operation, but the analysis does not reconstruct the whole base string because parts depend on dynamic values that we cannot determine statically. This can then result in an out-of-bounds exception, which would cause us to miss more values. For example, if a URL obtained dynamically would contain a 32 character device serial number, but our placeholder is `from_~pref`, then the analysis may cause an out-of-bounds exception if it accesses index 9. We mitigate this issue by preempting the calls that can cause such issues and expand the value on demand. Notably, this is not limited to string operations, but also extends to accessing arbitrary member fields of objects. For

missing parameters or base objects, we attempt to create them with their default constructors. For primitive data types (e.g., boolean, int, or float), we assign default values.

Local vs. Remote Endpoints ⑥. A unique aspect of companion apps is that their communication can be local, to connect to IoT devices or hubs, or remote, to connect to remote backends. We need to distinguish these classes to answer what data they share, how, with whom, and what the security and privacy impact is. Therefore, we categorize endpoints as certainly local or possibly remote. That is, we identify local connections by checking whether a reconstructed endpoint points to a local IP address, a broadcast address, a multicast address, or the domain originates from user input (`fromUI.local`). We consider all other endpoints as remote.

3.2 Data-Flow Analysis

In the second phase of IoTFlow, we use DFA to trace data flows from IoT devices and sensitive Android methods. IoTFlow builds on FlowDroid [12], which is a data-flow framework for Android. We extended it to address the unique challenges of the IoT ecosystem. We (1) connect reconstructed endpoint information to data sent or received, similar to pointer analysis, and (2) trace flows across ICC.

Considering how modern apps work internally, we must pay particular attention to ICC. It is now the recommended way for app components to communicate with each other and often used, which is why tracing data flow across it is crucial. Theoretically, FlowDroid supports ICC via ICCTA [56]. However, ICCTA cannot generate ICC models for current Android apps [66, 98], which prevents FlowDroid from tracing flows through ICC. IoTFlow addresses this blind spot by treating ICC as sources and sinks, and connecting an ICC sink (writing to a key) to the corresponding ICC sources (reading from the key) by reconstructing the key used in ICC through VSA.

Connecting Reconstructions ⑥. After reconstructing network endpoint information with VSA, we must connect them to the points where the app adds data to the request objects or receives a response, as these might be different from where the endpoint is set. For example the endpoint might be set during initialization of a connection object that is later used (repeatedly) to send or receive data. We identify the points where the app receives data and use the receiving statement as communication trigger points. Similarly, we need to connect a request's destination with the request's data when the request is executed. We do so using multiple data-flow analysis runs, which we split by method type for easier parallelization (e.g., MQTT, UDP, or CoAP). Returning to Listing 2, we previously reconstructed the MQTT broker endpoint via VSA (line 15). For our DFA in the next step, we now associate the MQTT broker endpoint (line 15) to the sink `publishString` (line 18) (marked \rightarrow).

Direct Data Flows (Source to Sink) ⑦. We are interested in data flows from sources that are (1) Bluetooth, (2) responses from the local network, or (3) sensitive Android methods. We trace them to (1) ICC sinks and (2) remote sinks, that is, data leaks.¹ Bluetooth data is interesting as it may contain data from smart devices and local network communication is likely data from smart devices.

Crucially, we need to treat flows to and from the same method differently depending on the context and how the app uses the

method (e.g., we want to analyze local network responses but ignore responses from remote endpoints). Thus, we extended FlowDroid to support context-sensitive flow analysis. We precisely identified the methods and the context that we need to consider as trigger points with the help of our VSA and by Connecting Reconstructions ⑥, which we can utilize to understand potential data leaks.

We focus first on three types of straight-forward immediate flows: (1) Bluetooth to network, (2) local network to network, and (3) sensitive data to network. Additionally, we trace sources to ICC sinks, to analyze flows across ICC, giving us three more flow types: (4) Bluetooth to ICC, (5) local network to ICC, and (6) sensitive data to ICC. Considering our example Listing 1, here, IoTFlow identifies the flow (marked \rightarrow) from the Bluetooth source `bcg` in line 3 via line 5 to line 6, where the data value is passed to the intent using the key `BLE_DATA` (reconstructed via VSA, marked $\leftarrow\rightarrow$).

Indirect Data Flows (Source to ICC to Sink) ⑧. Finally, we need to follow up on the flows we identified that have an ICC sink, to properly bridge the ICC boundary. We trace the additional flow type (7) ICC source to network sink, and then precisely connect the new flows with previously identified flows of types (4)–(6). This allows us to discover and analyze data leaks involving ICC. For our examples Listing 1 and Listing 2, based on Direct Data Flows ⑦, we identified a flow from Bluetooth to ICC using the key `BLE_DATA`. In Listing 2, using our indirect flow analysis, we now identify the flow (marked \rightarrow) from the ICC source `getStringExtra()` in line 7 to line 9 to line 17 to line 18, where the app sends the Bluetooth data to the MQTT broker. Last, we connect the new ICC to network flow to the previously identified Bluetooth to ICC flow leveraging the VSA reconstructed ICC keys, giving us the indirect data flow that crosses the ICC boundary from Bluetooth to ICC to network.

4 INSIGHTS INTO THE IOT ECOSYSTEM

We evaluate IoTFlow on 10,836 apps on an Ubuntu 20.04.6 machine with 48 physical CPU cores (96 cores with hyper-threading, 2x Intel(R) Xeon(R) Gold 6342 CPU) and 1,024 GiB RAM. We limit the memory for the analysis of each app to 150 GiB (`-Xmx150g`).

4.1 Dataset

Verified Companion Apps. We analyze IoTFlow on 9,889 unique IoT companion apps that were verified manually by prior work as part of three individual datasets [50, 62, 63]. We refer to our consolidated dataset as IoT-VER. It contains 455 apps collected by Neupane et al. [63] for studying if apps follow best practices, 5,100 apps that Jin et al. [50] used for the training, validation, and testing of IoTSpotter, and 6,208 apps that Nan et al. [62] collected and manually verified for IoTProfiler. Three quarters of the IoTProfiler apps are from the Google Play Store (74.6%), the remaining apps are from third-party stores. We did not augment these datasets with additional apps to not fragment the IoT companion app dataset space, which we deem important for reproducibility. Unfortunately, the public IoTSpotter dataset is incomplete and it misses 128 apps. Neupane et al.'s dataset misses two apps for which only the package name is available. We excluded these apps from our dataset.

All three datasets have 118 apps in common. IoTSpotter and IoTProfiler share 1,430 apps. The dataset of Neupane et al. shares 57 apps with IoTSpotter and 21 apps with IoTProfiler. If multiple

¹Full list of sources and sinks: <https://github.com/SecPriv/iotflow/tree/main/config>

datasets contain the same app, we only analyze the most recent version, that is, the app with the highest version code, since it is monotonically increasing [7]. Our consolidated dataset IoT-VER contains 9,889 apps, unique by their package names.

Popular General-purpose Apps. We also downloaded 1,000 popular apps and games from the *top selling free* category of the Google Play Store in January 2022, which we use to illustrate the differences between IoT companion apps and other apps. We manually removed companion apps from the dataset and refer to the remaining 947 apps as GP-2022. To do so, two researchers independently classified each app based on its metadata in the Google Play Store. If they disagreed, they studied it in-depth until they reached an agreement.

4.2 Performance

We first discuss the performance of IoTFlow on our datasets (see Table 1). In addition to the total run time, we investigate the required time separately for VSA and DFA. On average, general apps take almost five times as long to analyze as companion apps (125m31s vs. 26m23s). This difference is even more pronounced when considering the median (129m36s vs. 6m51s): The processing time for companion apps is almost 20x faster than for general apps. Reasons may be the larger code base of general-purpose apps or that they tend to have more sources and sinks that we need to consider. Overall, we consider a median analysis time of less than 7 minutes and an average analysis time of approximately 26 minutes practical.

VSA Performance. We allow up to 600 backward traces for each identified statement to prevent long-running analyses. Increasing the number of backward traces typically leads to more combinations of the same data, like request parameters. Each backward trace has up to 300 steps. We determined these thresholds empirically, observing a reasonable trade-off between resources and precision. Additionally, we configure timeouts for backward tracing (15 minutes) and forward computation (20 minutes). Our analysis only triggered the backward timeout when analyzing 11 (0.1%, all from GP-2022) apps and the forward timeout for 304 (2.81%; 155, 1.57% IoT-VER and 149, 15.73% GP-2022) apps, which we consider reasonable. Higher thresholds could lead to more flows being found.

Data-Flow Performance. For DFA, we increased the timeout suggestions by the FlowDroid authors [14] by 50%. We set the FlowDroid callback collection timeout to 7m30s and the timeout for flow analysis to 15m. Our analysis triggered the callback timeout for 2,432 apps (22.44%) and the flow analysis timeout for 3,004 apps (27.72%). Separating the two datasets, 1,847 companion apps (18.68%) and 585 general-purpose apps (61.77%) triggered the callback timeout, while 2,484 companion apps (25.12%) and 520 general-purpose apps (54.91%) triggered the flow analysis timeout.

4.3 How Companion Apps Communicate

To answer *RQ1: How do companion apps and devices communicate?*, we identify device-to-app communication and the involved network protocols, and we study certificate pinning.

4.3.1 Direct Device Communication. First, we analyze the reconstructed values for indicators of direct communication with the devices, such as local IP addresses, broadcast, and multicast addresses, user-configurable addresses (i.e., endpoints from user input; marked

Table 1: Dataset and Performance Overview. We show for the VSA, Flow Analysis, and the total time (VSA+Flow Analysis), the average time (Avg.), median time (Med.), and standard deviation (Std.) per app in minutes [minutes:seconds].

Dataset	# Apps	VSA			Flow Analysis			Total		
		Med.	Avg.	Std.	Med.	Avg.	Std.	Med.	Avg.	Std.
IoT-VER	9,889	1:52	5:40	9:46	3:59	21:19	31:29	6:51	26:23	37:20
GP-2022	947	75:53	70:44	36:40	55:57	54:47	40:29	129:36	125:31	65:56

Table 2: Number of Apps using Direct Device Communication. Indicators are hard-coded local network IP addresses (grouped if found in 30 or more apps), user-configurable addresses (fromUI.local), broadcast and multicast, or Bluetooth.

Address	IoT-VER	GP-2022
10***	716 (7.24%)	12 (1.27%)
10.0.0.172	516 (5.22%)	1 (0.11%)
10.0.0.200	438 (4.43%)	
10.10.2.2	48 (0.49%)	7 (0.74%)
other	242 (2.45%)	12 (1.27%)
172.16-31.*	103 (1.04%)	4 (0.42%)
172.17.0.1	49 (0.50%)	1 (0.11%)
other	56 (0.57%)	3 (0.32%)
192.168.*	746 (7.54%)	4 (0.42%)
192.168.0.1	115 (1.16%)	
192.168.1.1	180 (1.82%)	2 (0.21%)
192.168.1.3	36 (0.36%)	
192.168.4.1	77 (0.78%)	
other	518 (5.24%)	2 (0.21%)
fe80	3 (0.03%)	
Multicast and Broadcast	452 (4.57%)	4 (0.42%)
224.0.0.251	127 (1.28%)	1 (0.11%)
239.255.255.250	74 (0.75%)	
255.255.255.255	241 (2.44%)	4 (0.42%)
IPv4 other	93 (0.94%)	
IPv6 other	3 (0.03%)	
fromUI.local	123 (1.24%)	1 (0.11%)
Bluetooth	6,355 (64.26%)	180 (19.01%)

as fromUI.local), and Bluetooth permissions. The latter indicates that the devices themselves might not have Wi-Fi capabilities, but that they use the companion app as a gateway to access the Internet. Some devices may also spawn their own Wi-Fi network that the phone needs to join for pairing. Within the network, the device has a fixed address known by the companion app. The apps can also use broadcasts to discover devices in local networks, for example apps use Universal Plug and Play (UPnP) to find devices that support screen mirroring. A fourth method is asking the user directly. Table 2 summarizes our findings.

IoT-Verified. 6,355 (64.26%) apps declare at least one Bluetooth permission. We find a local IP address in 1,483 (14.99%) apps, a broadcast or multicast addresses in 452 (4.57%) apps, and addresses from user input in 123 (1.24%) apps. Among broadcast and multicast addresses, we found the broadcast address 255.255.255.255 (2.44%) most often, followed by the multicast DNS (mDNS) 224.0.0.251 (1.28%), and UPnP's 239.255.255.250 (0.75%). Besides the IPv4 addresses, we found three (0.03%) IPv6 multicast addresses.

General-purpose Apps. We observe a significant lower number for all four direct device communication indicators for general-purpose apps in GP-2022. We find local IP addresses in only 2.21% of apps, compared to 14.99% in IoT-VER. Similarly, broadcast and multicast addresses drop from 4.57% in IoT-VER to 0.42% in GP-2022. Only

one (0.11%) address depends on user input in GP-2022, compared to 123 (1.24%) addresses in IoT-VER. The number of apps requesting Bluetooth permissions also decreased from 64.26% in IoT-VER to 19.01% in GP-2022. These findings strengthen our assumption that our direct device communication indicators are indeed meaningful.

Takeaways. We identified four strategies apps use to communicate locally with smart devices, and we show by comparing them to general-purpose apps that they are indeed specific to companion apps. Identifying this kind of communication helps security and privacy analyses (see Section 4.5.2). Prompting the user for the device location and using multicast can be dangerous and prone to misconfigurations. Users might make devices unwittingly accessible over the Internet [18]. A Shodan [84] query for open port 554 returns 78,858 results of exposed cameras, suggesting that misconfigured devices accessible remotely are a common issue. Attackers can also sniff broadcast packages or mimic the legitimate device to act as a Monkey-in-the-Middle (MITM) [29]. Finally, we note that any information about local network devices is sensitive and can be abused for advertising and tracking purposes [52].

We recommend to use device discovery and avoid requiring user configurable addresses, to reduce the risk of accidental misconfigurations [30]. Apps should also respect users' privacy and not send local network information to remote servers. In fact, they should prefer local communication over cloud communication whenever possible, as remote requests can reveal usage patterns to others.

4.3.2 URL Protocol Schemes. We identify network protocols based on the values we reconstructed through VSA. First, we analyze the URL schemes of the endpoints that apps communicate. Second, as we reconstruct endpoint information for libraries for *AMQP*, *MQTT* and *XMPP* communication, we can draw conclusions about them, even if they do not use specific schemes. Table 3 summarizes our results. The row *IoT-related* summarizes the schemes and protocols that are tailored to IoT devices. We group IPP, IPPs, RMTP, and VNC as *IoT-other* as we found them only in one or two apps, and we group protocols from IANA's list of URI schemes [48] that are less interesting for our use case (e.g., *service*, *about*, *info*) as *Other*. Overall, for IoT-VER, we reconstructed schemes in 7,113 unique apps for remote endpoints and in 871 apps for local communication.

HTTP(S). We find that apps still widely use plain HTTP. Our numbers represent an upper bound as we do not know how many actual connections occur over HTTP since we base our results on statically reconstructed endpoints. In practice, HTTP might be upgraded by default, but even if used as a fallback, HTTP can lead to security and privacy issues through protocol downgrade attacks.

Our results show that a high proportion of HTTP traffic, compared to HTTPS traffic, is for local communication. This is not surprising: Local communication might appear safe, and deploying TLS properly for IoT devices remains challenging [72]. Nevertheless, even if communication is local, TLS protects against eavesdroppers, which is important as devices use broadcast media like Wi-Fi.

MQTT Endpoints. Smart devices have unique usage scenarios and requirements, such as device-to-device communication and energy efficiency. Traditional communication protocols do not satisfy these requirements. New protocols can fit these demands, but they can also threaten security and privacy, especially if they were designed

Table 3: Number of Apps with Reconstructed URL Protocol Schemes. Percentages are relative to the numbers of total apps with at least one scheme. For IoT-VER, we identified schemes for 871 local endpoints and 7,113 remote endpoints. For GP-2022, we identified schemes for 14 local endpoints and 898 remote endpoints. Protocols marked with a star (*) are based on IoTFlow identifying the corresponding libraries.

Protocol	Local		Possibly Remote	
	IoT-VER	GP-2022	IoT-VER	GP-2022
Android			29 (0.41%)	184 (20.49%)
File	4 (0.46%)		2,180 (30.65%)	578 (64.37%)
FTP	1 (0.11%)		8 (0.11%)	
HTTP	788 (90.47%)	13 (92.86%)	4,901 (68.90%)	639 (71.16%)
HTTPS	81 (9.30%)	2 (14.29%)	5,445 (76.55%)	885 (98.55%)
<i>IoT-related</i>	49 (5.63%)		315 (4.43%)	1 (0.11%)
<i>AMQP*</i>			6 (0.08%)	
<i>Cast</i>			4 (0.06%)	
<i>CoAP*</i>	2 (0.23%)		9 (0.13%)	
<i>CoAPs*</i>			2 (0.03%)	
<i>MQTT*</i>	27 (3.10%)		158 (2.22%)	1 (0.11%)
<i>Palm</i>			58 (0.82%)	
<i>RTSP</i>	1 (0.11%)		15 (0.21%)	
<i>RTSPs</i>			2 (0.03%)	
<i>TV</i>			9 (0.13%)	
<i>URN</i>	8 (0.92%)		18 (0.25%)	
<i>XMPP*</i>	11 (1.26%)		29 (0.41%)	1 (0.11%)
<i>IoT-other</i>			4 (0.06%)	
JAR			65 (0.91%)	1 (0.11%)
SMB	4 (0.46%)		62 (0.87%)	2 (0.22%)
WS	14 (1.61%)	7 (50.00%)	130 (1.83%)	10 (1.11%)
WSS	2 (0.23%)		138 (1.94%)	14 (1.56%)
<i>Other</i>	7 (0.80%)		1,604 (22.55%)	830 (92.43%)

without considering an adversarial environment or if developers make wrong assumptions about them. One protocol used often by IoT devices is the *Message Queuing Telemetry Transport (MQTT)* protocol. In practice, it often lacks authentication and authorization, allowing attackers to access user data or take over devices [49, 97].

MQTT is the most widespread IoT-specific communication protocol for IoT-VER. We reconstructed 147 MQTT endpoints in 176 apps, of which nine represent local IP addresses. We verify that the remaining 138 remote endpoints are indeed valid by opening a connection to them. To not raise any ethics concerns, we only open and immediately close the connection, and we do *not* perform any action (e.g., subscribing to a topic). We use the Python Paho library [31] for our test and base our results on the return code: If the connection is successfully established (return code 0) or an error related to connection parameters is returned (return codes 1 to 5), we consider the endpoint as reachable and valid.

We connected successfully (return code 0) to 74 MQTT endpoints (53.62%). To further investigate the remaining 64 endpoints, we probed for other ports typically used for MQTT (1883 and 8883) with *nmap* [59]. Seven endpoints were *closed* and 37 were *filtered*, meaning our connection attempts were prevented at the network level. One reason may be geographical restrictions. The remaining 20 endpoints were unresponsive to ICMP echo requests and we consider them unreachable.

MQTT Credentials. IoTFlow can also reconstruct authentication credentials. Hard-coding credentials into the app can lead to attacks on the integrity and confidentiality of data by allowing an attacker to connect and publish or subscribe to topics (e.g., modifying a parameter of a physical actuator). We reconstructed 30 unique usernames and 34 unique passwords in IoT companion apps.

MQTT Topics and Payloads. Our analysis can reconstruct the topics (i.e., topics for which the phone or IoT device should receive messages) and message payload formats (i.e., the format of messages shared between phone, IoT device, and the cloud). We found 726 topic names and 330 payload formats. While we may miss dynamic values from communication with the device, the information we gain is valuable to understand the behavior of IoT apps and devices.

Other IoT Protocols. We also identified other IoT protocols in IoT-VER apps, namely XMPP, AMQP, and CoAP. Among the 36 XMPP endpoints we identified, we could connect to five, the port was filtered for six, and the remaining 25 endpoints were unresponsive to ICMP echo requests. For the six identified AMQP endpoints, we could connect to two, we received an authentication error for one, and the AMQP-specific port was closed or filtered for the remaining endpoints. For the two CoAP endpoints, one was a local IP address, but we successfully reconstructed 55 unique URL paths used to specify the location of resources on the server.

General-purpose Apps. For GP-2022, we reconstructed local addresses only in combination with HTTP, HTTPS, and WS (Web-Socket). Like for companion apps, only a minority uses HTTPS locally (14.29%). However, unlike IoT apps, nearly all apps (98.55%) use it for remote communication. Unsurprisingly, general apps do not use IoT protocols. Only a card game app uses MQTT and XMPP.

Takeaways. We found widespread adoption of HTTP across IoT-VER apps despite its insecurity. For GP-2022 apps, the situation improves as almost all apps communicate over HTTPS. However, in both datasets, most local communication does not adopt TLS to secure the connection. We also identified how IoT-specific protocols (MQTT, AMQP, XMPP, CoAP) are being used and we reconstructed crucial information, like credentials and topics.

Generally, apps should not use hard-coded credentials but generate them individually during initialization, use limited and narrow authorization scopes, follow best practices (e.g., encrypting Android shared preferences [8]), and encrypt all communication (e.g., via TLS, but preferably end-to-end).

4.3.3 Pinning and Certificates. An additional aspect of how companion apps secure their communication is certificate pinning. It is a contentious topic: While OWASP [92] suggests it when the app wants to verify the host's identity, Google [3] advises not to adopt it because of issues deriving from certificate changes. However, determining whether it is good or bad is out of scope of our work.

We use the approach by Pradeep et al. [76] to identify pinning and the corresponding certificates by analyzing the Network Security Configuration (NSC) specified in the Android Manifest and the certificates included in the app. Table 4 summarizes our results.

IoT-Verified. More companion apps include certificates (12.21%) than use pinning (3.89%). On average, each app includes 3.21 certificates. More than half of the certificates in IoT-VER were self-signed, possibly to communicate with IoT devices. We also investigate if certificates were expired when the apps were downloaded. If the download date is unknown, we infer it based on the app versions. Our numbers are lower bounds for the apps from IoTProfiler and Neupane et al. because we assume apps were downloaded on the first day of the year when they could have been downloaded. We treat certificates as expired if their expiration date is before 2018

Table 4: Certificates and Pinning. The first rows show the number of apps in which we found pinning, certificates, and apps containing expired or self-signed certificates. The remaining rows show the corresponding certificates. The number of expired certificates at the time of download is a lower-bound for IoT-VER because it is not always known.

		IoT-VER	GP-2022
Apps	Pinning	385 (3.89%)	111 (11.72%)
	Certificates	1,207 (12.21%)	119 (12.57%)
	Expired (at download)	474 (4.79%)	49 (5.17%)
	Expired (May 2023)	822 (8.31%)	59 (6.23%)
	Self-Signed	1,042 (10.54%)	91 (9.61%)
Certificates	Total Number	31,285	1,837
	Expired (at download)	3,976 (12.71%)	268 (14.59%)
	Expired (May 2023)	9,129 (29.18%)	321 (17.47%)
	Self-Signed	18,018 (57.59%)	684 (37.23%)
	Avg per App (Std)	3.21 (20.97)	1.94 (14.59)

for IoTProfiler and before 2021 for apps by Neupane et al. Apps might be downloaded later, but this does not threaten validity as our numbers are a lower bound. For IoTSpotter apps, the download date is available as they were downloaded via AndroZoo [2]. Overall, 12.71% certificates were expired when the apps were downloaded (in 4.79% apps). In May 2023, 822 (8.31%) apps contain 9,129 (29.18%) expired certificates. Intuitively, expired certificates point to poor security practices and can even prevent communication.

General-purpose Apps. Compared to IoT-VER, more apps adopt pinning (11.72% vs. 3.89%), but the same proportion of apps include certificates (12.57% vs. 12.21%). On average, however, they include less certificates (1.94 vs. 3.21). One reason may be the lower number of self-signed certificates. While more than half of all (57.59%) certificates are self-signed for companion apps, only slightly more than one third (37.23%) of certificates are self-signed for general-purpose apps. At the download date, 14.59% certificates were already expired. IoT-VER's older download date could be a reason for the increase in expired certificates in May 2023 (29.18% vs. 17.47%).

Takeaways. Comparing included, expired, and self-signed certificates, we can conclude that more certificates do not lead to better security. Companion apps include substantially more certificates than general-purpose apps, and proportionally significantly more of them are expired or self-signed. Interestingly, fewer companion apps adopt the controversial practice of certificate pinning.

Generally, developers should renew certificates well before expiration, as users may not install updates immediately. Further care is needed for self-signed certificates as apps must add code to explicitly trust them, or Android will prevent the communication that attempts to use them. Worse, doing so incorrectly, like instructing TrustManager to trust every certificate, enables MITM attacks [3].

4.4 With Whom IoT Apps Communicate

After analyzing how apps communicate, we investigate *RQ2: Who are companion apps communicating with?* We categorize the reconstructed fully-qualified domain names (FQDNs) and effective top-level domains+1 (eTLD+1) to spot potentially problematic endpoints, like trackers, investigate where data is sent geographically, and analyze if endpoints are vulnerable to domain takeovers.

Table 5: Categorized Endpoints by IoTFlow for IoT-VER, GP-2022, and Comparison between IoTFlow (IF) and Dynamic Analysis (DA). We report with the number of unique FQDN per dataset and shared between them, prefixed with # the number of apps with at least one domain per category, the average number of domains per app, and the standard deviation.

	Large-scale IoTFlow Analysis						IoTFlow vs. Dynamic Analysis				
	IoT-VER	GP-2022	Shared	# IoT-VER	# GP-2022	Avg (SD) IoT	Avg (SD) GP	IF DA	\cap	\cap TLDs	# Apps
Advertisement	487	279	141	2,959 (29.92%)	848 (89.55%)	0.76 (1.73)	6.33 (4.92)	34 56	10 (12.50%)	9 (39.03%)	13 (100%)
Analytics	114	96	78	1,647 (16.65%)	679 (71.70%)	0.38 (1.03)	2.89 (3.68)	12 16	6 (27.27%)	5 (45.45%)	9 (69.23%)
CDNs	410	97	26	1,165 (11.78%)	733 (77.40%)	0.17 (0.64)	1.02 (0.97)	9 16	- (-)	- (-)	9 (69.23%)
Crash Reporting	4	3	3	195 (1.97%)	192 (20.27%)	0.02 (0.15)	0.23 (0.49)	6 1	1 (16.67%)	1 (50.0%)	3 (23.08%)
Social Networks	84	49	24	1,046 (10.58%)	137 (14.47%)	0.37 (1.45)	0.23 (0.79)	11 6	1 (6.25%)	1 (14.29%)	2 (15.38%)
Other	7,248	1,420	271	4,917 (49.72%)	685 (72.33%)	2.08 (4.57)	3.52 (5.29)	80 84	17 (11.56%)	19 (25.33%)	12 (92.31%)

4.4.1 Advertisers and Trackers. We classify the FQDNs to learn who receives data from the app and, via the app, from the devices. We use the domain lists by Ren et al. [82], which they compiled from various ad-blocking lists. Additionally, we use the Exodus tracker list [36]. Table 5 summarizes our results.

IoT-Verified. Overall, 2,959 (29.92%) apps include 487 unique advertisement FQDNs and 1,647 (16.65%) apps use 114 analytic-related FQDNs. Although they belong to different categories, both domains behave similarly by collecting user information. We also reconstructed 410 FQDNs pointing to Content Distribution Networks (CDNs) in 1,165 (11.78%) apps. Additionally, we identified 84 social network FQDNs shared across 1,046 (10.58%) apps, with the respective standard deviation indicating that if apps use one, they often use more. The remaining 7,248 FQDNs in 4,917 (49.72%) apps do not fall in our categories and we label them *Other*.

General-purpose Apps. The average number of advertisement and tracker FQDNs per general-purpose app is 6.33, eight times higher than per companion app (0.76). Additionally, they occur in almost all apps (89.55%), while they only occur in less than one third (29.92%) of companion apps. The situation for analytics FQDNs is similar (71.70% vs. 16.65%). Most analytics and crash reporting FQDNs are shared between the two datasets, while FQDNs from other categories are mainly limited to one dataset.

Takeaways. The large number of 7,248 *Other* FQDNs in companion apps combined with the low number of 271 FQDNs shared with general-purpose apps (3.25% of all IoT FQDNs) suggests that many are IoT-specific. Prior work observed a low coverage of existing filter lists for IoT domains [60, 88], highlighting the need for more scrutiny by future work into who receives data by these apps. Prior work showed that users value IoT security and privacy [32] and are willing to pay a premium for devices that respect their security and privacy [33]. Indeed, not using ad services or trackers could be a unique and convincing differentiating value proposition for IoT devices, especially because users already pay for the device.

4.4.2 Geographic Location. Next, we determined the location of the reconstructed FQDNs to study where data is sent and which countries receive IoT data. We first resolved the FQDNs to determine the location of the IPv4 addresses against the allocated blocks [34]. We resolved them from Vienna, Austria, which is in a jurisdiction that has implemented the EU's General Data Protection Regulation (GDPR) [35]. Notably, due to geographic split horizon DNS (GeoDNS), the resolved IP addresses may differ for other vantage points. Table 6 shows aggregated geographic regions. We

Table 6: Geographic Location of Network Endpoints. The numbers show the amount of endpoints from each location and the ratio to the overall number of endpoints.

	US	CN	EU	Asia	UK	RU	Other
IoT-VER	17,283 (46.09%)	10,221 (27.25%)	5,380 (14.35%)	3,166 (8.44%)	438 (1.17%)	113 (0.30%)	901 (2.40%)
GP-2022	10,606 (79.69%)	181 (1.36%)	1,786 (13.42%)	207 (1.56%)	47 (0.35%)	299 (2.25%)	183 (1.38%)

perform our analysis at the FQDN level because the FQDN endpoint receives the data. This granularity is also important because FQDN and eTLD+1 locations can differ. For example, xiaomi.com is hosted in China, but ru.register.xmpush.xiaomi.com is hosted in Russia.

We can make multiple observations comparing endpoint locations between companion apps and general-purpose apps. First, substantially fewer endpoints for companion apps are in the US (46.09%) than they are for general-purpose apps (79.69%). The difference (33.6 percentage points) stems almost exclusively from more Chinese endpoints (27.25% to 1.36%, 25.89 pp), with the remainder (7.71 pp) being nearly covered by other Asian countries for IoT-VER (8.44% compared to 1.56%, 6.88 pp). Other regions remain mainly stable.

Takeaways. The scattered geographic location of endpoints might raise privacy concerns. Countries have implemented various data protection regulations with stricter or more relaxed requirements. For example, the EU's GDPR [35] is considered the world's strongest privacy law. If a European user downloads an app that contacts endpoints outside the EU, their data is subject to GDPR, but the app may transfer it to foreign countries and process it there. This clearly raises privacy concerns and may even be illegal. Moreover, even if no sensitive data is sent directly, metadata can suffice to infer usage patterns, which can be sensitive (e.g., for smart locks).

4.4.3 Abandoned Domains. Domains that could be re-registered but are still used pose severe security and privacy risks for users as attackers could take them over. A similar argument applies to domains that are registered, but for which DNS information is stale and where the corresponding IP address could be taken over [17]. We focus on expired domains as they provide longer-term capabilities to attackers. We extract the eTLD+1 from the reconstructed FQDNs to identify abandoned domains. We then resolve the eTLD+1 to test whether they are in use. For domains we cannot resolve, we use WHOIS to check if it is registered or free.

IoT-Verified. We identified 136 potentially abandoned domains in companion apps. After manually investigating and removing artifacts, we verified that 67 domains from 73 apps are indeed

available for registration. They are in apps for watches, TVs, cars, health equipment, security and baby cameras, lights, and locks. An attacker could take over these devices by registering the domains.

We also investigated if the 73 apps can still be downloaded from the Google Play Store. Unavailable apps remain critical, but differently so. They can still impact users as the devices might not have been replaced and they might still connect to those domains. We found that 27 apps (37.0%) are available. Remarkably, one app has over one million downloads, a second app has over 500,000, and three others have more than 100,000. For ten apps, based on the reconstruction information, it is likely the domains receive IoT data. They use IoT information in URLs, such as `ipcDeviceIdList` as a request parameter, or `petInfoData/addpet` as a path. Eight apps use abandoned domains to download files, which may be executed or could be device updates. Sixteen domains are API endpoints and also likely receive sensitive data. We responsibly disclosed our findings to developers and the Google Play Store.

Takeaways. Pariwono et al. [73] investigated abandoned domains for general apps, but the dangers can be more serious for IoT devices. Attackers could not only take over the apps and receive PII, but they might also be able to control hundreds of thousands of devices, enabling large-scale distributed denial-of-service attacks and allowing them to create botnets. Our analysis shows (1) that abandoned domains are a real danger in the IoT ecosystem, (2) that they affect a varied range of devices, and (3) what data they receive.

Developers should actively monitor the domains that their apps may contact, including those of third-party libraries. Additionally, old or deprecated domains that may still be contacted should remain registered, as users may depend on outdated app versions, and made inoperable instead of allowing others to register it.

4.5 What Data Companion Apps Share

To answer RQ3: *Which data are companion apps sharing (and how)?*, we first report what data apps can access, based on the requested permissions, to understand what data they could share. We then analyze the data flows we extracted to identify leaked data and whether encryption is used to protect data.

4.5.1 Permissions. We extract permissions and *protectionLevel* with Androguard [28]. We focus on permissions with a *protectionLevel* of *dangerous* (permissions protecting sensitive resources) and *privileged* (permissions that third-party apps should not adopt).

On average, GP-2022 apps request more permissions than companion apps (17.54, SD 14.10 vs. 14.26, SD 10.23). For IoT-VER, 8,769 (88.67%) apps request at least one dangerous permission. `WRITE_EXTERNAL_STORAGE` occurs most often (7,209 IoT apps, 72.9% and 559 general-purpose apps, 59.03%). The second and third most frequent permissions are `ACCESS_COARSE_LOCATION` (5,735 IoT apps, 57.99%) and `ACCESS_FINE_LOCATION` (5,529 IoT apps, 55.91%) as in most cases IoT devices also rely on location to perform their functions (e.g., smart watches recording physical activity).

Additionally, 2,604 (26.33%) IoT apps request one or more privileged permissions, while only 73 (7.71%) GP-2022 apps do. Even if system permissions are requested, they will only be granted if the phone is rooted or if the app has a special entitlement (e.g., the phone vendor may grant such an entitlement to their own apps, and they might also produce IoT devices). They can also occur for

backward compatibility reasons or be remnants from development that were never removed (e.g., the second most common privileged permission is `READ_LOGS`, which appears in 998 IoT apps).

Finally, 5,660 (57.24%) IoT apps use “non-standard” permissions. The permission occurring the most belongs to Google Cloud Messaging (GCM) (3,656 apps, 36.97%) and is used when receiving a broadcast from GCM. We also find permissions of specific brands, for example, Huawei (60 permissions occur 1,207 times in 424 apps) or Sony (31 permissions 787 times in 424 apps).

Takeaways. General-purpose apps request more permissions than companion apps on average. However, IoT apps use more *privileged* and *dangerous* permissions, with two of the most requested *dangerous* permissions being for the user’s geographic location.

We recommend to regularly review if permissions are still current, to request the least necessary set of permissions, and to only temporarily acquire them when needed. With new Android updates, permissions might also change, for example, scanning for Bluetooth devices required location permissions only up to Android 12 [6, 42].

4.5.2 Data Flows. To learn more about what data is sent, we analyze the flows we discovered via DFA. Table 7 summarize our findings. We distinguish between three flow types based on the destination: Bluetooth, local network, or a sensitive Android API. We determine where the data is sent by connecting the VSA results with the individual flows. Unfortunately, we may not have precise information for all flows for two reasons: (1) VSA might not reconstruct an endpoint precisely, for example, because it depends on dynamic values, (2) we could not connect reconstruction and flow.

We identified data flows from Bluetooth and local network sources only for IoT apps, which is not surprising, as we have shown that such communications are companion app specific (see Section 4.3.1). Overall, we found 579 flows from Bluetooth sources in 90 apps. Remarkably, 497 (85.84%) of these flows involve ICC, which highlights the need for DFA that is ICC aware, like our approach. We precisely identified endpoint information (i.e., where the data is sent to) for 50 (8.64%) flows. For local network sources, we discovered 75 flows, of which four (5.33%) involve ICC. IoTFlow reconstructed precise endpoint information for 49 (65.33%) of them. Finally, we identified 6,706 flows from sensitive Android API in 1,682 (17.01%) IoT apps, and 1,366 such flows in 318 (33.58%) GP-2022 apps.

Case Study: Smart Grill. Our analysis finds a flow in a companion app for smart grills. The app reads data from the device via BLE, parses it, process it via an intent, and later sends it to an Amazon AWS endpoint via MQTT. We successfully connected to the endpoint without requiring credentials (anonymously) (see Section 4.3.2). This means that we could potentially receive data from others (for ethical reasons, we did not explore this further).

Case Study: Smart Camera. In a smart camera companion app, we found a flow from `getDeviceId` to a remote endpoint. The app uses the IMEI together with a username and password for authentication. Worth mentioning is also that the app hashes the password using MD5, which is insecure and cryptographically broken. Afterward, the app encrypts the username and password with 3DES, which is also insecure and cryptographically broken. IoTFlow’s VSA reconstructed the key, even though the app developers put one byte

Table 7: Flow Analysis. We separated the flows by their categories. The ICC-Flow columns represent the flows involving any ICC, and the endpoint columns the flows with additional endpoint information. The ratio concerns the number of flows from the category. The app columns show the number of apps with the respective flows and the relation to the apps in the dataset.

Dataset	Bluetooth				Local Network				Android			
	ICC-Flows	Endpoints	Flows	Apps	ICC-Flows	Endpoints	Flows	Apps	ICC-Flows	Endpoints	Flows	Apps
IoT-VER	497 (85.84%)	50 (8.64%)	579	90 (0.91%)	4 (5.33%)	49 (65.33%)	75	53 (0.54%)	2,340 (34.89%)	1,952 (29.11%)	6,706	1,682 (17.01%)
GP-2022									420 (30.75%)	619 (45.31%)	1,366	318 (33.58%)

of the key into a different class file, potentially trying to obfuscate it and avoid regex-based key recovery.

Sharing the IMEI is problematic because users can only change the IMEI by physically replacing the device as it is a non-resettable hardware identifier. Google strongly discourages developers from using any hardware identifiers, including the IMEI [5], and it is also prohibited by Android’s user data policy [40]. With Android 10 (API level 29, released in 2019), Google added additional restrictions to access the IMEI [5, 9], but around 14.4% of users are still using older versions, allowing apps to access these identifiers [15].



Geographic Location. We also analyze the geographic location of data flows with endpoint information. For IoT-VER, we find 917 (15.04%) flows sending data to Chinese and 604 (9.90%) to US endpoints. The share of flows with US endpoints increases for general-purpose apps (75, 27.88%), while the share for Chinese drops (12, 4.46%), aligned with their distribution (see Section 4.4.2). Positively, as we conducted our experiments from the EU, most destinations are within the EU: 73.80% for IoT-VER and 67.66% for GP. Unfortunately, it also means that more than 25% of destinations are outside the EU, potentially violating GDPR. The situation is worse for Bluetooth-based sources than it is for local network flows. For Bluetooth, 27 endpoints (45.76%) are US endpoints, and for local network flows, 37 endpoints (52.86%) are Chinese endpoints. Our artifact provides more details.²










Takeaways. With the help of IoTFlow’s combination of VSA and DFA, we identified real-world security and privacy issues in the IoT ecosystem and we discussed what data companion apps leak and where they send it, which we illustrated with two examples.

Following best practices and for privacy reasons, developers should minimize data they collect and use, and only send data if it is truly necessary. Generally, we recommend to process as much data as possible locally, and to encrypt any data leaving the devices.

4.5.3 Encryption Analysis. Finally, we analyze the encryption algorithms apps use and we investigate the reconstructed data passed to cryptographic methods. We reconstructed the algorithms in 812 (85.74%) GP-2022 apps and in 4,069 (41.15%) IoT-VER apps. Table 8 summarizes our results. AES is the most widely used encryption algorithm for IoT apps (92.97%) and GP-2022 apps (99.38%). Algorithms that are considered insecure or cryptographically broken are much more prominent in IoT apps (1,461 apps, 35.80%) using encryption than they are in GP-2022 apps (135, 16.63%).

We also evaluate reconstructed encryption keys. Unfortunately, removing false positive artifacts is extremely challenging because it is difficult to determine whether a key is truly used as is. For example, a 16-byte array with all 0 values could be an insecure

Table 8: Encryption Algorithms. The number of apps that use the respective encryption algorithm and its relation to the number of apps with encryption algorithms (4,069 IoT-VER, and 812 in GP-2022). Recommended algorithms are marked . Algorithms considered insecure or broken are marked .

Algorithm	IoT-VER	GP-2022
 AES	3,794 (92.97%)	807 (99.38%)
 ChaCha	4 (0.10%)	3 (0.37%)
 Diffie-Hellman	14 (0.34%)	44 (5.42%)
 RSA	16 (0.39%)	
Serpent	136 (3.33%)	31 (3.82%)
 Blowfish	79 (1.94%)	10 (1.23%)
 DES	1,366 (33.47%)	120 (14.78%)
 3DES	351 (8.60%)	66 (8.13%)
 GOST	5 (0.62%)	
 RC4	288 (7.06%)	21 (2.59%)

key, or it may not have been initialized. Therefore our numbers are an upper bound. Overall, we reconstructed hard-coded keys in 2,321 (57.04%) IoT apps and 408 (50.24%) general-purpose apps.

Takeaways. The main differences between IoT apps and GP-2022 apps are in what encryption they use and how they use them. Using hard-coded keys and broken encryption algorithms gives a false sense of security and does not provide security or privacy. Unfortunately, both issues are worse for companion apps.

Beyond using strong encryption algorithms, we also recommend to initialize encryption keys on demand and to store them securely, for example, with the help of Android KeyStore [4].

5 IOTFLOW VS. DYNAMIC ANALYSIS

Our static analysis approach has some limitations, especially because we do not require access to the device. It is crucial to understand what and how much information we can truly reconstruct from companion apps without the device. We verify the accuracy and completeness of the reconstructed values by analyzing and comparing the results we obtained through IoTFlow with our in-depth manual analysis when interacting with apps and devices. To this end, we recorded traffic when using 13 different devices and their companion apps in our lab environment (see Table 9).

Our test environment uses a machine running Ubuntu 20.04 with frida-tools [78] and mitmproxy [26], and a rooted Google Pixel 4 running frida-server [78] on Android 12. The machine hosts a Wi-Fi network to which the phone and the devices connect, providing Internet connectivity through an Ethernet connection. We bypass certificate pinning via Frida’s built-in scripting. We test companion apps with two strategies: First, *automatic inputs*, we test apps with the Application Exerciser Monkey (AME) [10] for 10 minutes or until they crash, whichever occurs first. We do not

²<https://github.com/SecPriv/iotflow/tree/main/scripts/evaluation/dfa>

Table 9: Tested Devices. The IoT devices that we tested dynamically together with their device type and package name.

Device	Type	Package Name
Bose QC35	Headphones	com.bose.monet
Divoom Timebox	Alarm Clock	com.divoom.Divoom
Fitbit Inspire 1	Smart Watch	com.fitbit.FitbitMobile
Blaupunkt	Smart Watch	cn.xiaofengkj.fitpro
HHCC FlowerCare	Plant Sensor	com.huahuacaoao.flowercare
Hama WiFi	Light Bulb	com.hama.smart
Philips Hue	Light Bulb	com.signify.hue.blue
Ikea DIRIGERA	Smart Hub	com.ikea.tradfri.lighting
Anti-Lost	Smart Tracker	com.lenztech.kindelf
LIFX A60	Light Bulb	com.lifx.lifx
Nut Find3	Smart Tracker	com.nut.blehunter
Soundcore Life Q35	Headphones	com.oceanwing.soundcore
Wiz Colour	Light Bulb	com.tao.wiz

expect to trigger complex behavior of IoT devices (e.g., because it would require us to set up the device), but AME remains a common testing technique [22] and we include it for completeness. Second, *manual inputs*, we manually interact with each app for 30 minutes and trigger all functionalities, including pairing and interacting with IoT devices, changing their settings, etc. Indeed, we observed significantly less traffic with AME than with manual interaction, demonstrating the scalability issues of dynamic analysis.

From the observed traffic, we extract requests' domain names, which correspond to who receive data, and resource paths, which correspond to functionality (e.g., API endpoints). We match them based on exact string equivalence exactly between IoTFlow and dynamic analysis. Considering the configuration of our dynamic environment, we also manually match domains and paths by (a) identifying over-approximate placeholders, such as the device product code, serial number, etc. and matching them to concrete dynamic information, (b) generalizing the dynamic system configuration, like language and locale, (c) grouping repeated dynamic data (as they also do not provide new information in the dynamic analysis setting, but provide a false sense of accuracy), and (d) resolving network-level redirects (e.g., DNS-based or IP anycast). We remove analysis artifacts that are *clearly not* related: (a) domains and resources that were requested from outside of the IoT app, such as by the Android operating system (e.g., background update checks), (b) domains and resources that were requested by Android WebView components unrelated to device behavior (e.g., opening a vendor's online shop website), and (c) invalid domain names. We retain all data that cannot be clearly attributed to dynamic analysis artifacts, making our results a lower-bound. Last, as we focus on IoT-related behavior, we manually label data as related if it relates to IoT device behavior, security, privacy, or data exchange. Our artifact provides further details on the identified domains and paths, and their matching.³

IoTFlow extracted 214 domains from the 13 companion apps, with a minimum of 3 domains, an average of 16.46 domains, and a maximum of 42 domains per app. With dynamic analysis, we observed 218 domains, with a minimum of 1 domain, an average of 16.77 domains, and a maximum of 48 domains per app. Between static analysis and dynamic analysis, 36 domains match exactly and we matched 7 additional domains manually.

³https://github.com/SecPriv/iotflow/tree/main/dynamic_analysis

We categorize all domains using our previous approach (see Section 4.4). Table 5 summarizes our results and our artifact provides further details.³ Notably, we find substantially more advertisement domains via dynamic analysis than through IoTFlow. This is expected because of how modern ads are targeted and auctioned, requiring dynamic information. IoTFlow only recovers the entry point for ads, but this is actually sufficient to determine that they are used. It also highlights an important issue: Considering all domains gives a false sense of accuracy toward dynamic analysis, many of which may not provide new insight. For example, while it confirms our findings of extensive tracking in IoT apps, the IKEA companion app contacts 48 domains in total, but it also contacted 20 advertising domains and four social network domains.

Focusing on *certainly* IoT domains, IoTFlow and dynamic analysis share 21 domains across all apps (min 0, avg 1.62, max 5), IoTFlow identified 33 domains that dynamic analysis missed (0/2.54/10), and dynamic analysis found 19 unique domains (0/1.46/4). That is, IoTFlow performs better than or equal to dynamic analysis for 9/13 devices and worse for only 4/13 devices (Fitbit smart watch, Hama light bulb, Soundcore headphones, and Wiz light bulb). For 2/4 of these apps, Fitbit and Wiz, IoTFlow correctly identifies the effective TLD of all domains we observed dynamically, that is, the operator, but it missed some subdomains. For Hama, it misses four IoT endpoints that we saw dynamically, likely because the device is a rebranded IoT device. For Soundcore, it misses one dynamically generated domain pointing to the device's most recent firmware.

Beyond domains, we also compare requests' paths. It allows us to assess which approach is more promising to comprehensively understand IoT device behavior, meaning if one discovers more IoT-related functionality or if they identify distinct (overlapping) sets of behavior. Both approaches identified the same 50 IoT-related paths over all 13 apps (min 0, avg 3.85, max 17). We statically identified an additional 231 IoT-related paths (min 0, avg 17.77, max 45) and 496 general paths (min 2, avg 38.15, max 77). Dynamic analysis found 110 unique IoT-related paths (min 0, avg 8.46, max 32) and 337 general paths (min 1, avg 25.92, max 54). For three apps (Fitbit, Hue, and Wiz), our static analysis performs worse. Fitbit and Wiz use annotations to construct paths, which we cannot analyze, a limitation we share with state of the art (see Section 6). For Hue, our approach extracts 2 IoT-related path, while we observe 3 paths dynamically. IoTFlow performs better or equal for 10/13 apps, with a factor of at least 1.14x (Divoom, 41 vs. 36) and up to 31x (Flowercare, 31 vs. 1). For the IKEA app, dynamic analysis did not find any IoT-related paths, while IoTFlow found 45 paths.

Overall, IoTFlow performs better than dynamic analysis and extracts more IoT-related behavior statically from companion apps than dynamic analysis (54 domains and 281 paths vs. 40 domains and 160 paths) for most apps (9/13), it performs comparable for one app, and it performs slightly worse for the remaining apps (3/13).

IoTFlow Findings. Taking an in-depth look into IoTFlow's security and privacy findings for the 13 apps, we find that:

- 8/13 apps send information via unencrypted HTTP to third parties, which an attacker could eavesdrop on or modify (e.g., if they are on the network path or the same wireless network). If unencrypted data is used to configure or update the device, then taking over control could be possible [68]. The NUT Find3 item

tracker retrieves notifications over unencrypted HTTP, which can allow an attacker to modify a user's notification (e.g., to show that a lost item was found and where or that it has moved away).

- 5/13 apps use hard-coded symmetric encryption keys (e.g., for AES), which allows attackers to eavesdrop on their communication and can allow them to impersonate the remote end (e.g., to push configurations or updates by extracting the keys from the companion app) [67, 69].
- 2/13 apps send the hardware identifiers (IMEI) to countries outside of the EU, that is, outside of the GDPR region (one might send it to Russia and one to China to a remote endpoint that indicates tracking), using an API that is deprecated (see Section 4.5.2).
- 5/13 apps, while less critical, use country-level location information and send this to remote endpoints.
- No apps use hard-coded authentication credentials, but this does not imply that they are secure because they might not use any authentication at all.

6 LIMITATIONS AND FUTURE WORK

IoTFlow has limitations inherent to static analyses. Additionally, we utilize the existing frameworks Soot and FlowDroid, and we inherit their limitations. For example, our resilience to obfuscation is limited, which can affect signature-based identification of sources and sinks. We find that they are only a minor share for companion apps (2.66% obfuscated). However, they are more prevalent for general-purpose apps (10.81% obfuscated) based on an APKid [37] analysis of our datasets. Obfuscation is also an orthogonal problem, and new deobfuscation techniques can readily be adopted. Similarly, we focus our analysis on the Dalvik bytecode of apps, that is, we do not support native code. We currently do not consider code annotations, which the *retrofit* library uses to specify request paths and network methods. Both techniques are infrequently used, and not tackling them is a limitation we share with prior work, as existing frameworks struggle to support them, and the required engineering effort to support them is substantial.

Our DFA supports ICC, but our VSA does not. We plan to extend ICC support to VSA in future work. Only 2.01% of reconstructed values contain ICC data, which does not invalidate our results. Currently, we limit ICC tracking to the same app, but theoretically, ICC can cross app boundaries or come from websites via deep links, providing further avenues for collusion.

For our analysis, we limit the number of backward steps and set a timeout, which trades between precision and resources but could lead to missing values and flows. We empirically determined our thresholds and other limits could yield more precise results.

Motivated by our results on certificate pinning and abandoned domains, we aim to study how companion apps evolve over time. Naturally, identifying network endpoints, protocols, and APIs is only the first step toward truly understanding the security and privacy of device-to-cloud communication in the IoT ecosystem.

7 RELATED WORK

Following, we compare IoTFlow to related work in IoT security, IoT companion app analysis, and general-purpose app analysis.

IoT Security. Prior work in IoT security largely focused on identifying attacks on a small set of devices. Wood et al. [96] investigated

how medical IoT devices communicate and transmit data, and they found them leaking information, like the measurement frequency, despite using encryption. Chu et al. [23] discovered kids' devices sending PII over unencrypted connections. Other work [13, 20, 38, 65, 93] focused on Samsung's SmartThings apps. SmartThings is a smart hub ecosystem unifying the control of compatible devices and allows event flow graphs, which are conceptually simple [64]. In contrast, IoTFlow analyzes arbitrary Android apps, which are more widespread and significantly more complex. Correspondingly, their techniques do not readily transfer to the entire IoT ecosystem. Related work also investigated the ecosystem via crowd-sourced network traffic collection [46] or telemetry data [53] of real-world user devices, which raises ethical and anonymization challenges.

IoT Companion App Analysis. Different work investigated IoT companion apps *in combination with* physical devices [21, 61, 80, 81] to find security and privacy issues. For example, Zhou et al. [100] studied the interactions between IoT devices, cloud, and apps using state machines and found issues that can lead to device hijacking. However, they require the IoT devices, which prevents scalability. We overcome this limitation with our new static analysis approach design. Wang et al. [94] analyzed companion apps without the corresponding device. Instead of analyzing and determining *how* and *with whom* the apps communicate, as we do, they focused on identifying re-branding and propagation of known vulnerabilities. That is, they require prior domain knowledge about other devices and existing vulnerabilities. Similarly, Jin et al. [50] aimed to identify companion apps at scale, and then to identify known vulnerabilities in the apps, such as outdated library versions. Nan et al. [62] analyzed IoT apps with machine learning to detect code that handles IoT-related data, and then assessed whether the behavior was communicated to the user. Naturally, their statistical machine learning approach fundamentally differs from our static program analysis approach. Other work [85, 86, 99, 102] aims to find BLE issues in mobile apps. IoTFlow is more general as we investigate communication beyond BLE, thus obtaining a better understanding of a greater part of the IoT ecosystem.

General-purpose App Analysis. Understanding general-purpose mobile apps has seen significant work. Some approaches use dynamic analysis to run apps in controlled environments to observe their (network) behavior and endpoints [25, 57, 58, 75, 79, 82]. As we observed, the provided inputs impact the analysis, which is an ongoing research challenge [19, 22, 43]. Moreover, to adopt these approaches, one would need the actual IoT devices, making large-scale analysis infeasible. Several approaches extract information about apps through static analysis techniques, like VSA or flow analysis, such as network endpoints, API keys, protocol commands, etc. Gadiant et al. [39] extract URLs and JSON schemas to study HTTP(S) usage, private APIs, and code injection vulnerabilities. Extractocol [51] reconstructs HTTP requests based on data flow analysis for automated protocol analysis, but does not scale. Stringoid [77] simulates string concatenations, but cannot reconstruct URLs built in other ways, such as *okhttp3.HttpUrl.Builder*. Leakscope [101] reconstructs API keys in mobile apps. Zuo et al. [102] reconstructed BLE UUIDs to identify vulnerable implementations of its pairing process. Wen et al. [95] reconstructed Controller Area Network (CAN) bus commands.

8 CONCLUSION

We introduced IoTFlow, a new technique for the large-scale security and privacy analysis of IoT devices through their companion apps. With Value Set Analysis (VSA), we extract network endpoints and protocols, which enables us to characterize IoT device behavior for local app-to-device communication and remote communication with cloud backends without requiring the physical IoT device. By cleverly combining VSA with Data-flow Analysis (DFA), we trace data flows from IoT devices and sensitive Android methods to understand better what data companion apps share and how. Leveraging IoTFlow, we analyzed 9,889 companion apps and 947 general-purpose apps. We identified striking differences between the two types of apps and discovered various security and privacy problems in the IoT ecosystem, such as abandoned domains, hard-coded credentials, expired certificates, or use of broken encryption algorithms. Our approach shows clear promise for identifying security and privacy issues of IoT devices at scale and it could be used to generate privacy labels or verify claimed behavior automatically.

ACKNOWLEDGMENTS

We thank Andrea Continella for providing valuable feedback on an early draft of this paper. This material is based on research supported by the Vienna Science and Technology Fund (WTF) and the City of Vienna through projects ICT19-056 and ICT22-060, the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972, and SBA Research (SBA-K1), a COMET center funded by BMK, BMDW, and the state of Vienna.

REFERENCES

- [1] N. Alexopoulos, M. Brack, J. P. Wagner, T. Grube, and M. Mühlhäuser. "How Long Do Vulnerabilities Live in the Code? A Large-Scale Empirical Measurement Study on FOSS Vulnerability Lifetimes". In: *31st USENIX Security Symposium (USENIX Security)*. Aug. 2022. K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon. "AndroZoo: Collecting Millions of Android Apps for the Research Community". In: *13th International Conference on Mining Software Repositories (MSR)*. May 2016. doi: [10.1145/2901739.2903508](https://doi.org/10.1145/2901739.2903508).
- [2] Android Developers. *Security with network protocols*. Oct. 27, 2021. URL: <https://developer.android.com/training/articles/security-ssl> (visited on 10/07/2022).
- [3] Android Developers. *Android Keystore system*. Oct. 7, 2022. URL: <https://developer.android.com/training/articles/keystore> (visited on 07/25/2023).
- [4] Android Developers. *Best practices for unique identifiers*. May 22, 2023. URL: <https://developer.android.com/training/articles/user-data-ids> (visited on 07/25/2023).
- [5] Android Developers. *Bluetooth permissions*. July 28, 2023. URL: <https://developer.android.com/guide/topics/connectivity/bluetooth/permissions> (visited on 07/28/2023).
- [6] Android Developers. *Version your app*. Apr. 12, 2023. URL: <https://developer.android.com/studio/publish/versioning> (visited on 04/27/2023).
- [7] Android Developers. *Work with data more securely*. July 12, 2023. URL: <https://developer.android.com/topic/security/data> (visited on 07/25/2023).
- [8] Android Developers. *TelephonyManager*. URL: <https://developer.android.com/reference/android/telephony/TelephonyManager> (visited on 07/13/2022).
- [9] Android Developers. *UI/Application Exerciser Monkey*. URL: <https://developer.android.com/studio/test/other-testing-tools/monkey> (visited on 03/30/2022).
- [10] AppBrain. *Android network libraries*. URL: <https://www.appbrain.com/stats/libraries/tag/network/android-network-libraries> (visited on 09/19/2021).
- [11] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Oetean, and P. McDaniel. "FlowDroid: Precise Context, Flow, Field, Object-Sensitive and Lifecycle-Aware Taint Analysis for Android Apps". In: *35th ACM SIGPLAN Conference on Programming Language Design and Implementation*. June 2014. doi: [10.1145/2594291.2594299](https://doi.org/10.1145/2594291.2594299).
- [12] L. Babun, Z. Berkay Celik, P. McDaniel, and S. Uluagac. "Real-time Analysis of Privacy-(un)aware IoT Applications". In: *21st Privacy Enhancing Technologies Symposium (PETS)*. July 2021. doi: [10.2478/popets-2021-0009](https://doi.org/10.2478/popets-2021-0009).
- [13] M. Balossini and S. Arzt. *GitHub FlowDroid Issue #578 – Flowdroid execution time*. Feb. 16, 2023. URL: <https://github.com/secure-software-engineering/FlowDroid/issues/578> (visited on 04/27/2023).
- [14] E. Belinski. *Android API Levels*. Sept. 4, 2023. URL: <https://apilevels.com/> (visited on 09/08/2023).
- [15] D. Boffey. *EU Recalls Children's Smartwatch over Data Fears*. Feb. 5, 2019. URL: <https://www.theguardian.com/technology/2019/feb/05/eu-recalls-childrens-smartwatch-over-data-fears> (visited on 01/30/2022).
- [16] K. Borgolte, T. Fiebig, S. Hao, C. Kruegel, and G. Vigna. "Cloud Strife: Mitigating the Security Risks of Domain-Validated Certificates". In: *25th Network and Distributed System Security Symposium (NDSS)*. Feb. 2018. doi: [10.14722/ndss.2018.23327](https://doi.org/10.14722/ndss.2018.23327).
- [17] K. Borgolte, S. Hao, T. Fiebig, and G. Vigna. "Enumerating Active IPv6 Hosts for Large-scale Security Scans via DNSSEC-signed Reverse Zones". In: *39th IEEE Symposium on Security & Privacy (S&P)*. May 2018. doi: [10.1109/SP.2018.00027](https://doi.org/10.1109/SP.2018.00027).
- [18] P. Carter, C. Mulliner, M. Lindorfer, W. Robertson, and E. Kirda. "CuriousDroid: Automated User Interface Interaction for Android Application Analysis Sandboxes". In: *International Conference on Financial Cryptography and Data Security (FC)*. Feb. 2016. doi: [10.1007/978-3-662-54970-4_13](https://doi.org/10.1007/978-3-662-54970-4_13).
- [19] Z. B. Celik, L. Babun, A. K. Sikder, H. Aksu, G. Tan, P. McDaniel, and A. S. Uluagac. "Sensitive Information Tracking in Commodity IoT". In: *27th USENIX Security Symposium (USENIX Security)*. Aug. 2018. doi: [10.5555/3277203.3277329](https://doi.org/10.5555/3277203.3277329).
- [20] J. Chen, W. Diao, Q. Zhao, C. Zuo, Z. Lin, X. Wang, W. 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](https://doi.org/10.14722/ndss.2018.23159).
- [21] S. R. Choudhary, A. Gorla, and A. Orso. "Automated Test Input Generation for Android: Are We There Yet?". In: *30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. Nov. 2015. doi: [10.1109/ASE.2015.89](https://doi.org/10.1109/ASE.2015.89).
- [22] G. Chu, N. Aphorpe, and N. Feamster. "Security and Privacy Analyses of Internet of Things Children's Toys". In: *IEEE Internet of Things Journal* 6.1 (Aug. 2019). doi: [10.1109/JIOT.2018.2866423](https://doi.org/10.1109/JIOT.2018.2866423).
- [23] K. Chung. *Taking over a Dead IoT Company*. Jan. 9, 2023. URL: <https://blog.kchung.co/taking-over-a-dead-iot-company/> (visited on 02/05/2023).
- [24] 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. doi: [10.14722/ndss.2017.23465](https://doi.org/10.14722/ndss.2017.23465).
- [25] A. Cortesi, D. Weinstein, D. Freed, T. Kriecheauer, P. F. Tirenna, M. Hils, and U. Verma. *mitmproxy - an interactive HTTPS proxy*. v6.0.2. Jan. 21, 2021. URL: <https://mitmproxy.org/> (visited on 01/15/2022).
- [26] A. Costin, J. Zaddach, A. Francillon, and D. Balzarotti. "A Large-scale Analysis of the Security of Embedded Firmwares". In: *23rd USENIX Security Symposium (USENIX Security)*. Aug. 2014. doi: [10.5555/2671225.2671232](https://doi.org/10.5555/2671225.2671232).
- [27] A. Desnos, G. Gueguen, S. Bachmann, and contributors. *GitHub – Androguard - Android Permissions*. July 3, 2020. URL: https://github.com/androguard/androguard/tree/master/androguard/core/api_specific_resources/aosp_permissions (visited on 05/02/2023).
- [28] F. Dhia and M. Dacier. "Zero Conf Protocols and Their Numerous Man in the Middle (MITM) Attacks". In: *15th IEEE Workshop on Offensive Technologies (WOOT)*. May 2021. doi: [10.1109/SPW53761.2021.00060](https://doi.org/10.1109/SPW53761.2021.00060).
- [29] C. Dietrich, K. Krombholz, K. Borgolte, and T. Fiebig. "Investigating Operators' Perspective on Security Misconfigurations". In: *25th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. Oct. 2018. doi: [10.1145/3243734.3243794](https://doi.org/10.1145/3243734.3243794).
- [30] Eclipse Foundation. *Paho - Python Client*. v1.5.1. Sept. 22, 2022. URL: <https://www.eclipse.org/paho/index.php?page=clients/python/index.php> (visited on 10/08/2022).
- [31] P. Emami-Naeini, J. Dheenadhayalan, Y. Agarwal, and L. F. Cranor. "Which Privacy and Security Attributes Most Impact Consumers' Risk Perception and Willingness to Purchase IoT Devices?". In: *42nd IEEE Symposium on Security & Privacy (S&P)*. May 2021. doi: [10.1109/SP40001.2021.00112](https://doi.org/10.1109/SP40001.2021.00112).
- [32] P. Emami-Naeini, J. Dheenadhayalan, Y. Agarwal, and L. F. Cranor. "Are Consumers Willing to Pay for Security and Privacy of IoT Devices?". In: *32nd USENIX Security Symposium (USENIX Security)*. Aug. 2023. (Visited on 08/21/2023).
- [33] I. Erben. *Country CIDR IP Ranges*. URL: <http://www.iwik.org/ipcountry/> (visited on 01/25/2022).
- [34] European Parliament and the Council of the European Union. "Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)". In: *Official Journal of the European Union* 59 (L19 May 4, 2016). URL: <http://data.europa.eu/eli/reg/2016/679/oj> (visited on 07/05/2021).
- [35] Exodus. *Trackers*. URL: <https://reports.exodus-privacy.eu.org/en/trackers/> (visited on 02/23/2022).
- [36] C. Fenton and contributors. *GitHub – APKID*. Nov. 12, 2020. URL: <https://github.com/rednaga/APKID> (visited on 01/29/2022).
- [37] E. Fernandes, J. Jung, and A. Prakash. "Security Analysis of Emerging Smart Home Applications". In: *1st IEEE European Symposium on Security & Privacy (EuroS&P)*. Mar. 2016. doi: [10.1109/SP.2016.44](https://doi.org/10.1109/SP.2016.44).
- [38] P. Gadiet, M. Ghafari, M.-A. Tarnutzer, and O. Nierstrasz. "Web APIs in Android through the Lens of Security". In: *27th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. Feb. 2020. doi: [10.1109/SANER48275.2020.9054850](https://doi.org/10.1109/SANER48275.2020.9054850).
- [39] Google. *User Data*. URL: <https://support.google.com/goopleplay/android-developer/answer/10144311> (visited on 07/25/2023).
- [40] C. Han, I. Reyes, Á. Feal, J. Reardon, P. Wijesekera, N. Vallina-Rodriguez, A. Elazari, K. A. Bamberger, and S. Egelman. "The Price is (Not) Right: Comparing Privacy in Free and Paid Apps". In: *20th Privacy Enhancing Technologies Symposium (PETS)*. July 2020. doi: [10.2478/popets-2020-0050](https://doi.org/10.2478/popets-2020-0050).
- [41] haxrob. *Twitter - HaxRob - Lightbulb App*. July 5, 2023. URL: <https://twitter.com/haxrob/status/1676424071452708864> (visited on 07/25/2023).
- [42] Y. He, L. Zhang, Z. Yang, Y. Cao, K. Lian, S. Li, W. Yang, Z. Zhang, M. Yang, Y. Zhang, and H. Huan. "TextExerciser: Feedback-driven Text Input Exercising for Android Applications". In: *41st IEEE Symposium on Security & Privacy (S&P)*. May 2020. doi: [10.1109/SP40000.2020.00071](https://doi.org/10.1109/SP40000.2020.00071).
- [43] G. Ho, D. Leung, P. Mishra, A. Hosseini, D. Song, and D. Wagner. "Smart Locks: Lessons for Securing Commodity Internet of Things Devices". In: *11th ACM ASIA Conference on Computer and Communications Security (ASIACCS)*. Apr. 2016. doi: [10.1145/2897845.2897886](https://doi.org/10.1145/2897845.2897886).
- [44] A. Holst. *IoT Connected Devices Worldwide 2019-2030*. Oct. 19, 2021. URL: <https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/> (visited on 01/15/2022).
- [45] D. Huang, N. Aphorpe, G. Acar, F. Li, and N. Feamster. "IoT Inspector: Crowdsourcing Labeled Network Traffic from Smart Home Devices at Scale". In: *ACM International Joint Conference on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT/Ubicomp)*. June 2020. doi: [10.1145/3397333](https://doi.org/10.1145/3397333).

- [47] IFTTT. *If This Then That – Connect Your Apps*. URL: <https://ifttt.com/> (visited on 07/24/2023).
- [48] Internet Assigned Numbers Authority (IANA). *Uniform Resource Identifier (URI) Schemes*. Jan. 28, 2022. URL: <https://www.iana.org/assignments/uri-schemes/uri-schemes.xhtml> (visited on 01/30/2022).
- [49] 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](https://doi.org/10.1109/SP40000.2020.00051).
- [50] X. Jin, S. Manandhar, K. Kafle, Z. Lin, and A. Nadkarni. "Understanding IoT Security from a Market-Scale Perspective". In: *29th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. Nov. 2022. doi: [10.1145/3548606.3560640](https://doi.org/10.1145/3548606.3560640).
- [51] J. Kim, H. Choi, H. Namkung, W. Choi, H. Choi, H. Hong, Y. Kim, J. Lee, and D. Han. "Enabling Automatic Protocol Behavior Analysis for Android Applications". In: *12th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*. Dec. 2016. doi: [10.1145/2999572.2999596](https://doi.org/10.1145/2999572.2999596).
- [52] D. Kuchhal and F. Li. "Knock and Talk: Investigating Local Network Communications on Websites". In: *21st Internet Measurement Conference (IMC)*. Nov. 2021. doi: [10.1145/3487552.3487857](https://doi.org/10.1145/3487552.3487857).
- [53] D. Kumar, K. Shen, B. Case, D. Garg, G. Alperovich, D. Kuznetsov, R. Gupta, and Z. Durumeric. "All Things Considered: An Analysis of IoT Devices on Home Networks". In: *28th USENIX Security Symposium (USENIX Security)*. Aug. 2019. doi: [10.5555/3361338.3361419](https://doi.org/10.5555/3361338.3361419).
- [54] P. Lamkin. *Report Claims Ring Employees Had Unfettered Access To Security Camera Footage*. Jan. 11, 2019. URL: <https://www.forbes.com/sites/paullamkin/2019/01/11/report-claims-ring-employees-had-unfettered-access-to-security-camera-footage/> (visited on 07/30/2021).
- [55] F. Li and V. Paxson. "A Large-Scale Empirical Study of Security Patches". In: *24th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. Oct. 2017. doi: [10.1145/3133956.3134072](https://doi.org/10.1145/3133956.3134072).
- [56] L. Li, A. Bartel, T. F. Bissyandé, J. Klein, Y. Le Traon, S. Arzt, S. Rasthofer, E. Bodden, D. Oeteanu, and P. McDaniel. "IcTA: Detecting Inter-Component Privacy Leaks in Android Apps". In: *37th IEEE/ACM International Conference on Software Engineering (ICSE)*. May 2015. doi: [10.1109/ICSE.2015.48](https://doi.org/10.1109/ICSE.2015.48).
- [57] M. Lindorfer, M. Neugschwandtner, and C. Platzer. "Marvin: Efficient and Comprehensive Mobile App Classification Through Static and Dynamic Analysis". In: *Annual International Computers, Software & Applications Conference (COMPSAC)*. July 2015. doi: [10.1109/COMPSAC.2015.103](https://doi.org/10.1109/COMPSAC.2015.103).
- [58] M. Lindorfer, M. Neugschwandtner, L. Weichselbaum, V. Fratantonio, V. van der Veen, and C. Platzer. "Andrubis - 1,000,000 Apps Later: A View on Current Android Malware Behaviors". In: *International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*. Sept. 2014. doi: [10.1109/BADGERS.2014.7](https://doi.org/10.1109/BADGERS.2014.7).
- [59] G. Lyon. *Nmap: the Network Mapper*. v7.80. Aug. 10, 2019. URL: <https://nmap.org/> (visited on 03/30/2022).
- [60] 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](https://doi.org/10.2478/popets-2021-0075).
- [61] D. Mauro Junior, 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](https://doi.org/10.1109/SPW.2019.00042).
- [62] Y. Nan, X. Wang, L. Xing, X. Liao, R. Wu, J. Wu, Y. Zhang, and X. Wang. "Are You Spying on Me? Large-Scale Analysis on IoT Data Exposure through Companion Apps". In: *32nd USENIX Security Symposium (USENIX Security)*. Aug. 2023.
- [63] S. Neupane, F. Tazi, U. Paudel, F. V. Baez, M. Adamjee, L. De Carli, S. Das, and I. Ray. "On the Data Privacy, Security, and Risk Postures of IoT Mobile Companion Apps". In: *36th IFIP Annual Conference on Data and Applications Security and Privacy (DBSec)*. July 2022. doi: [10.1007/978-3-031-10684-2_10](https://doi.org/10.1007/978-3-031-10684-2_10).
- [64] J. L. Newcomb, S. Chandra, J.-B. Jeannin, C. Schlesinger, and M. Sridharan. "IOTA: a calculus for internet of things automation". In: *ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward!)*. Oct. 2017. doi: [10.1145/3133850.3133860](https://doi.org/10.1145/3133850.3133860).
- [65] D. T. Nguyen, C. Song, Z. Qian, S. V. Krishnamurthy, E. J. M. Colbert, and P. McDaniel. "IoTSan: Fortifying the Safety of IoT Systems". In: *14th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*. Dec. 2018. doi: [10.1145/3281411.3281440](https://doi.org/10.1145/3281411.3281440).
- [66] NicolasFNino and S. Arzt. *GitHub FlowDroid Issue #601 – FlowDroid with EPICC*. Apr. 5, 2023. URL: <https://github.com/secure-software-engineering/FlowDroid/issues/601> (visited on 04/27/2023).
- [67] NIST. *CVE-2017-8866*. Dec. 11, 2017. URL: <https://nvd.nist.gov/vuln/detail/CVE-2017-8866> (visited on 07/26/2023).
- [68] NIST. *CVE-2019-16732*. Dec. 13, 2019. URL: <https://nvd.nist.gov/vuln/detail/CVE-2019-16732> (visited on 07/26/2023).
- [69] NIST. *CVE-2022-30271*. Dec. 13, 2022. URL: <https://nvd.nist.gov/vuln/detail/CVE-2022-30271> (visited on 07/26/2023).
- [70] F. R. Olivera. *Maven Repository*. URL: <https://mvnrepository.com/> (visited on 04/05/2023).
- [71] B. Pan. *GitHub – Dex2jar*. v2.1. Oct. 29, 2021. URL: <https://github.com/pxb1988/dex2jar> (visited on 01/15/2022).
- [72] M. T. Paracha, D. J. Dubois, N. Vallina-Rodriguez, and D. Choffnes. "IoTLS: Understanding TLS Usage in Consumer IoT Devices". In: *21st Internet Measurement Conference (IMC)*. Nov. 2021. doi: [10.1145/3487552.3487830](https://doi.org/10.1145/3487552.3487830).
- [73] E. Pariwono, D. Chiba, M. Akiyama, and T. Mori. "Don't Throw Me Away: Threats Caused by the Abandoned Internet Resources Used by Android Apps". In: *13th ACM ASIA Conference on Computer and Communications Security (ASIACCS)*. June 2018. doi: [10.1145/3196494.3196554](https://doi.org/10.1145/3196494.3196554).
- [74] S. Pletinckx, K. Borgolte, and T. Fiebig. "Out of Sight, Out of Mind: Detecting Orphaned Web Pages at Internet-Scale". In: *28th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. Nov. 2021. doi: [10.1145/3460120.3485367](https://doi.org/10.1145/3460120.3485367).
- [75] A. Pradeep, A. Feal, J. Gamba, A. Rao, M. Lindorfer, N. Vallina-Rodriguez, and D. Choffnes. "Not Your Average App: A Large-scale Privacy Analysis of Android Browsers". In: *23rd Privacy Enhancing Technologies Symposium (PETS)*. July 2023. doi: [10.56553/popets-2023-0003](https://doi.org/10.56553/popets-2023-0003).
- [76] A. Pradeep, M. T. Paracha, P. Bhowmick, A. Davanian, A. Razaghpahan, 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](https://doi.org/10.1145/3517745.3561439).
- [77] M. Rapoport, P. Suter, E. Wittern, O. Lhotak, and J. Dolby. "Who You Gonna Call? Analyzing Web Requests in Android Applications". In: *14th International Conference on Mining Software Repositories (MSR)*. May 2017. doi: [10.1109/MSR.2017.11](https://doi.org/10.1109/MSR.2017.11).
- [78] O. A. V. Ravnäs. *Frida*. v15.1.14. Dec. 9, 2021. URL: <https://frida.re/> (visited on 07/26/2023).
- [79] A. Razaghpahan, 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](https://doi.org/10.14722/ndss.2018.23009).
- [80] 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 Underconstrained Inputs for IoT Devices". In: *40th IEEE Symposium on Security & Privacy (S&P)*. May 2019. doi: [10.1109/SP40001.2021.000066](https://doi.org/10.1109/SP40001.2021.000066).
- [81] J. Ren, D. J. Dubois, D. Choffnes, A. M. Mandalari, R. Kolcun, and H. Haddadi. "Information Exposure From Consumer IoT Devices: A Multidimensional, Network-Informed Measurement Approach". In: *19th Internet Measurement Conference (IMC)*. Oct. 2019. doi: [10.1145/3355369.3355577](https://doi.org/10.1145/3355369.3355577).
- [82] 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](https://doi.org/10.14722/ndss.2018.23143).
- [83] S. J. Saidi, S. Matic, O. Gasser, G. Smaragdakis, and A. Feldmann. "Deep Dive into the IoT Backend Ecosystem". In: *22nd Internet Measurement Conference (IMC)*. Oct. 2022. doi: [10.1145/3517745.3561431](https://doi.org/10.1145/3517745.3561431).
- [84] Shodan. *Shodan Search*. URL: https://www.shodan.io/search?query=has_screenshot:true+port:554 (visited on 07/18/2022).
- [85] P. Sivakumaran and J. Blasco. "A Study of the Feasibility of Co-located App Attacks against BLE and a Large-Scale Analysis of the Current Application-Layer Security Landscape". In: *28th USENIX Security Symposium (USENIX Security)*. Aug. 2019. doi: [10.5555/3361338.3361340](https://doi.org/10.5555/3361338.3361340).
- [86] P. Sivakumaran, C. Zuo, Z. Lin, and J. Blasco. "Uncovering Vulnerabilities of Bluetooth Low Energy IoT from Companion Mobile Apps with Ble-Guide". In: *18th ACM ASIA Conference on Computer and Communications Security (ASIACCS)*. July 2023. doi: [10.1145/3579856.3595806](https://doi.org/10.1145/3579856.3595806).
- [87] V. Sivaraman, D. Chan, D. Earl, and R. Boreli. "Smart-Phones Attacking Smart-Homes". In: *9th ACM Conference on Security & Privacy in Wireless and Mobile Networks (WISEC)*. July 2016. doi: [10.1145/2939918.2939925](https://doi.org/10.1145/2939918.2939925).
- [88] C. Tagliaro, F. Hahn, R. Sepe, A. Aceti, and M. Lindorfer. "I Still Know What You Watched Last Sunday: Privacy of the HbbTV Protocol in the European Smart TV Landscape". In: *30th Network and Distributed System Security Symposium (NDSS)*. Feb. 2023. doi: [10.14722/ndss.2023.24102](https://doi.org/10.14722/ndss.2023.24102).
- [89] R. Vallée-Rai, E. Gagnon, L. Hendren, P. Lam, P. Pominville, and V. Sundaresan. "Optimizing Java Bytecode Using the Soot Framework: Is It Feasible?". In: *9th Compiler Construction*. Jan. 2000. doi: [10.1007/3-540-46423-9_2](https://doi.org/10.1007/3-540-46423-9_2).
- [90] N. Vallina-Rodriguez, J. Shah, A. Finamore, Y. Grunenberger, K. Papagiannaki, H. Haddadi, and J. Crowcroft. "Breaking for Commercial: Characterizing Mobile Advertising". In: *12th Internet Measurement Conference (IMC)*. Nov. 2012. doi: [10.1145/2398776.2398812](https://doi.org/10.1145/2398776.2398812).
- [91] J. Varmarken, H. Le, A. Shuba, A. Markopolou, and Z. Shafiq. "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](https://doi.org/10.2478/popets-2020-0021).
- [92] J. Walton, J. Steven, J. Manico, K. Wall, R. Iramar, and contributors. *Certificate and Public Key Pinning*. URL: https://owasp.org/www-community/controls/Certificate_and_Public_Key_Pinning (visited on 10/07/2022).
- [93] Q. Wang, W. Ul Hassan, A. Bates, and C. A. Gunter. "Fear and Logging in the Internet of Things". In: *25th Network and Distributed System Security Symposium (NDSS)*. Feb. 2018. doi: [10.14722/ndss.2018.23282](https://doi.org/10.14722/ndss.2018.23282).
- [94] 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](https://doi.org/10.5555/3361338.3361418).
- [95] H. Wen, Q. Zhao, Q. A. Chen, and Z. Lin. "Automated Cross-Platform Reverse Engineering of CAN Bus Commands from Mobile Apps". In: *27th USENIX Network and Distributed System Security Symposium (NDSS)*. Feb. 2020. doi: [10.14722/ndss.2020.24231](https://doi.org/10.14722/ndss.2020.24231).
- [96] D. Wood, N. Aporthe, and N. Feamster. "Cleartext Data Transmissions in Consumer IoT Medical Devices". In: *Workshop on Internet of Things Security and Privacy (IoT-S&P)*. Nov. 2017. doi: [10.1145/3139937.3139939](https://doi.org/10.1145/3139937.3139939).
- [97] H. Xu, M. Yu, Y. Wang, Y. Liu, Q. Hou, Z. Ma, H. Duan, J. Zhuge, and B. Liu. "Trampoline Over the Air: Breaking in IoT Devices Through MQTT Brokers". In: *7th IEEE European Symposium on Security & Privacy (EuroS&P)*. June 2022. doi: [10.1109/EuroS&P53844.2022.00019](https://doi.org/10.1109/EuroS&P53844.2022.00019).
- [98] yokotayokota, S. Arzt, and J. Samhi. *GitHub FlowDroid Issue #386 – Usage of IccTA in FlowDroid*. Sept. 13, 2021. URL: <https://github.com/secure-software-engineering/FlowDroid/issues/386> (visited on 04/27/2023).
- [99] Q. Zhao, C. Zuo, J. Blasco, and Z. Lin. "PeriScope: Comprehensive Vulnerability Analysis of Mobile App-defined Bluetooth Peripherals". In: *17th ACM ASIA Conference on Computer and Communications Security (ASIACCS)*. May 2022. doi: [10.1145/3488932.3517410](https://doi.org/10.1145/3488932.3517410).
- [100] 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](https://doi.org/10.5555/3361338.3361417).
- [101] 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](https://doi.org/10.1109/SP.2019.00009).
- [102] C. Zuo, H. Wen, Z. Lin, and Y. Zhang. "Automatic Fingerprinting of Vulnerable BLE IoT Devices with Static UUIDs from Mobile Apps". In: *26th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. Nov. 2019. doi: [10.1145/3319535.3354240](https://doi.org/10.1145/3319535.3354240).