AoT - Attack on Things: A security analysis of IoT firmware updates

Muhammad Ibrahim
Purdue University
West Lafayette, USA
ibrahi23@purdue.edu

Andrea Continella
University of Twente
Enschede, Netherlands
a.continella@utwente.nl

Antonio Bianchi
Purdue University
West Lafayette, USA
antoniob@purdue.edu

***Abstract*—IoT devices implement firmware update mechanisms to fix security issues and deploy new features. These mechanisms are often triggered and mediated by mobile companion apps running on the users' smartphones. While it is crucial to update devices, these mechanisms may cause critical security flaws if they are not implemented correctly. Given their relevance, in this paper, we perform a systematic security analysis of the firmware update mechanisms adopted by IoT devices via their companion apps. First, we define a threat model for IoT firmware updates, and we categorize the different potential security issues affecting them. Then, we analyze 23 popular IoT devices (and corresponding companion apps) to identify vulnerable devices and the SDKs that such devices use to implement the update functionality. Our analysis reveals that 6 popular SDKs present dangerous security flaws. Additionally, we fingerprint each vulnerable SDK and we leverage our fingerprints to perform a large-scale analysis of companion apps from the Google Play Store. Our results show that 61 popular devices and 1,356 apps rely on vulnerable SDKs, thus, they potentially adopt an insecure firmware update mechanism.**

1. Introduction

The adoption of IoT devices has been increasing constantly over the past few years. According to a study by Avast [12], the number of IoT devices will triple by 2025, reaching over 75 billion devices. As a result, an increasing number of IoT devices are showing up in different settings, ranging from corporate networks [20] to short-term rentals such as Airbnb [49]. Thus, researchers are actively studying the security and privacy concerns imposed by the adoption of these devices [27], [28], [30], [40], [41], [43]–[45], [51], [67], [71], [74], [76], [80], [83]–[85], [88].

Normally, IoT devices make use of a relay device, in most cases a smartphone, to accept commands from users. To provide an interface for user interaction, a mobile app is often installed on the smartphone to communicate with the IoT device. These apps are commonly referred to as **companion apps** [56], and they utilize diverse types of channels and protocols for handling communication with the IoT devices (e.g., Bluetooth or Wi-Fi).

A critical feature for the security of IoT devices is their firmware update mechanism, which allows for fixing bugs, improving security, or adding new features. The mechanism for sending firmware updates is referred to as 'device firmware update' or **DFU**. Often, IoT device vendors make use of companion apps, running on smartphones, to send firmware updates to the IoT devices. Naturally, if the DFU mechanisms are not securely implemented, IoT devices can miss critical security patches or can be compromised by executing malicious code.

Previous works [10], [23], [34], [47], [69] identified specific vulnerabilities in the firmware update mechanisms of some IoT devices. However, the state-of-the-art lacks a comprehensive and systematic picture of DFU issues in the IoT ecosystem. In fact, existing works only focus on a few selected products from specific vendors and do not provide a scalable categorization approach. Besides, the previously investigated attacks require access to the hardware of the IoT devices, significantly limiting the scope to attack scenarios that include physical access.

Conversely, in this paper, we examine mobile companion apps and their critical role in triggering, controlling, and mediating the updating of IoT devices. Specifically, we first investigate the app-mediated DFU mechanisms of IoT devices, understanding their threat model and potential vulnerabilities. We then present a comprehensive methodology for identifying vulnerable firmware update mechanisms in IoT devices on a large scale. A high-level overview of our methodology is illustrated in Figure 1, which includes three components: *AoT-Scout*, a manual systematic analysis to study DFU mechanisms of IoT Devices via their companion apps; *Attack-Tester*, a systematic manual testing procedure to test DFU mechanisms of IoT Devices against firmware attacks; *App-Rumbling*, an automated pipeline for classifying companion apps based on the vulnerable SDKs used for DFU.

Methodology. We systematically investigate the threat model of IoT firmware update mechanisms and define the following three categories of attacks that can be performed via their companion apps:

- **Firmware Modification Attack** (ModAttack) This attack allows an attacker to inject untrusted, malicious firmware into the target IoT device.
- **Firmware Downgrade Attack** (DownAttack) This attack allows an attacker to downgrade the firmware of the target IoT device.
- **Device Bricking Attack** (BrickAttack) This attack allows an attacker to render the IoT device unusable.

The root cause of these attacks is the fact that cryptographic verification and rollback prevention of the firmware updates is absent or incorrectly implemented. Our goal is to detect the presence of these vulnerabilities in the firmware update mechanisms of IoT devices at a large scale. To achieve this goal, we aim to utilize fingerprints of vulnerable companion app SDKs for a large-scale analysis of Android marketplaces.

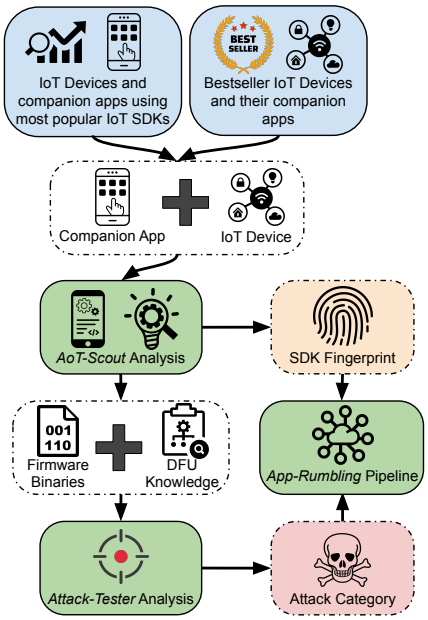

Figure 1: Overview of our methodology.

We collect a dataset of IoT devices and their companion apps using the following two sources:

1) *Amazon Bestseller IoT Devices*: We collect the Bestseller IoT devices and their companion apps from Amazon Bestseller products [8].
2) *Top SDKs used in companion apps*: We collect the companion apps (and corresponding devices) using popular SDKs identified by IoTSpotter, a prior study of IoT devices and companion apps [42].

To test this dataset, we first follow the *AoT-Scout* analysis to obtain both information about the update mechanism and the firmware binary. Using the extracted knowledge, we then use *Attack-Tester* to test companion apps for the presence of DFU vulnerabilities. Specifically, following the *Attack-Tester* analysis, we test which of the aforementioned attacks can be performed by transferring modified firmware binaries to the target IoT device.

Additionally, *AoT-Scout* enables us to identify, for each vulnerable device, the SDK used to implement the DFU mechanism. In fact, companion apps typically use dedicated libraries (i.e., SDKs) to interact with their corresponding IoT devices and to implement the DFU functionality. Our analysis reveals 6 vulnerable SDKs. We confirm these vulnerabilities by exploiting, in our lab, 8 devices that use the identified vulnerable SDKs.

Finally, for each of these vulnerable SDKs, we generate a code fingerprint. By leveraging the vulnerable SDKs' fingerprints, we assess the extent of vulnerable SDK usage in the companion apps on the Android app marketplaces.

Our results show that SDKs vulnerable to firmware attacks are widely used in the companion apps of IoT devices. In particular, we find that 61 popular devices and 1,356 apps rely on vulnerable SDKs, thus, they potentially adopt an insecure firmware update mechanism.

Contributions. In summary, in this paper we make the following contributions:

- We investigate the state of firmware update mechanisms adopted by IoT devices and their potential vulnerabilities.
- We propose two manual approaches (*Attack-Tester* and *AoT-Scout*) to study and detect vulnerable firmware update mechanisms of 23 IoT devices.
- We identify that 6 popular SDKs used by IoT companion apps to implement the DFU functionality are vulnerable to the studied attacks.
- We generate fingerprints of the vulnerable SDKs, and we use them to perform the first large-scale IoT DFU vulnerability analysis using our *App-Rumbling* pipeline. Our analysis reveals 1,356 apps (and 61 popular devices) using these vulnerable SDKs.

2. Background

2.1. IoT Devices

As a definition of IoT device, we use the following definition: *An IoT device is a physical device that can communicate data over a network.* However, in this paper, we restrict our scope to IoT devices with certain characteristics to ease our analysis. We explain our scope of IoT devices below.

As Chen et al. [19] shows that majority of the IoT Devices communicate with a companion iOS or Android app. There might be some IoT devices that **only** communicate with non-Android relay devices such as iOS devices. We exclude those kinds of IoT devices from our study to ease our analysis.

We also exclude IoT devices that are not bare metal devices. As discussed by Salehi et al. [58], a bare metal is a device that is not running any operating system and the majority of the IoT devices are bare metal devices. We exclude non-bare metal devices from our study because our study involves performing firmware attacks via companion apps and most non-bare metal devices do not perform firmware updates via their companion apps.

Consumer IoT devices are mostly controlled using a relay device such as a smartphone. IoT device vendors publish companion mobile apps for their IoT devices. The companion apps provide an interface for the end users to interact with the IoT devices. The IoT device vendors employ a variety of mechanisms to facilitate communication between the IoT device and its companion app. The two main methods of communication between an IoT device and its companion app are: Bluetooth and Wi-Fi.

While using Bluetooth, the IoT device and the smartphone communicate directly with each other. Since Bluetooth supports short-range communication, the devices must be in physical proximity.

While using Wi-Fi, the IoT device and the smartphone usually do not communicate directly. A common mechanism is to rely on an MQTT [31] broker that acts as a proxy for communication between the IoT device and the companion app. The user sends commands using the companion app to the MQTT broker, which forwards them to the respective IoT device.

In the next section, we discuss how these communication mechanisms are used to perform firmware updates.

2.2. Companion Apps

Companion apps are the primary interface for the end users to interact with the IoT devices. Companion apps are

installed on relay devices, such as smartphones, and can communicate with their corresponding IoT devices using a variety of mechanisms, as mentioned in the previous section. Companion apps are used to perform various tasks such as device configuration, monitoring, and firmware updates. These tasks are handled by SDKs (Software Development Kits) which are present in the companion apps. SDKs provide a set of APIs that can be used by the developers to perform a variety of tasks such as firmware updates without having to write the code from scratch.

2.3. IoT Device Firmware Update (DFU)

IoT device vendors use companion apps to perform firmware updates on their IoT devices. We will use the term DFU to refer to Device Firmware Update. DFUs are necessary to fix bugs and add new features to the IoT devices. DFUs involve transferring an updated firmware binary to the IoT device. A *firmware repository* stores the updated firmware binaries of the IoT devices. The firmware repository can be the companion app itself or a separate backend server. The first step of the DFU process is to check if there is updated firmware available in the firmware repository. Different IoT vendors can implement the DFU mechanism in different ways to facilitate the firmware updates of their IoT devices. However, the basic procedure for checking if a DFU is available can be generalized as follows:

1) The companion app requests the current firmware version from the IoT device.
2) After receiving the current firmware version, the companion app checks if there is a newer firmware version available in the firmware repository.
3) If a newer firmware version is available, the updated firmware is transferred to the IoT device.

The implementation of transferring the updated firmware binary depends on the type of communication used by the IoT device. We now discuss the basic DFU binary transfer procedures when Bluetooth or Wi-Fi communication is used.

When using Bluetooth, the firmware binary is directly transferred from the companion app to the IoT device. To perform DFU, the companion app needs to have the updated firmware binary. In this case, the firmware repository can be the companion app itself or a separate backend server. If the companion app itself is the firmware repository, the IoT device vendor packages the updated firmware binary within the companion app and publishes the app on the Android marketplaces. The companion app with the updated firmware binary is downloaded and installed on the smartphone. Then, it checks if an updated firmware is available and if so, the updated firmware binary is transferred directly using Bluetooth to the IoT device. If the firmware repository is a separate backend server, the companion app requests the latest firmware version from the backend server. If an updated firmware version is available, the companion app downloads the updated firmware binary to the smartphone. Finally, the updated firmware binary is transferred to the IoT device using Bluetooth.

When using Wi-Fi, an MQTT broker and a backend firmware repository are involved in the transfer of the firmware binary to the IoT device. Generally, the following steps (illustrated in Figure 2) are performed for transferring the firmware binary:

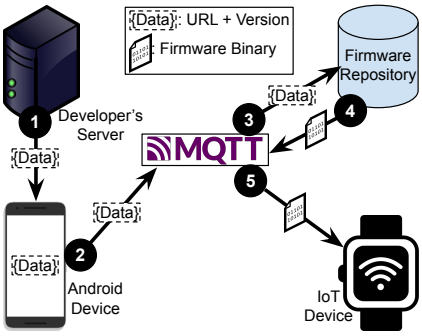

Figure 2: Wi-Fi IoT DFU involving an MQTT broker. The MQTT broker serves as proxy for communication between the IoT device and the companion app. 'Data' contains information about the firmware binary. Specifically, the latest firmware version and firmware's URL.

1) The developer's server sends the URL of the firmware repository server and the version of the updated firmware binary to the companion app.
2) The companion app checks if an updated version of the firmware is available and sends the URL of the firmware to the MQTT broker.
3) The MQTT broker sends a request to fetch the firmware from the repository using the URL.
4) The MQTT broker downloads the firmware image from the firmware repository.
5) The firmware image is transferred from the MQTT broker to the IoT device.

Within this general communication framework, there can be device-specific variations in the steps of the MQTT DFU mechanism. For example, in Step 3, the URL can be sent to the IoT device, which can then directly download the firmware from the firmware repository.

3. Firmware Attacks on IoT

3.1. Threat Model

To perform the firmware attacks, we assume the attacker has the capability to intercept and modify the communication between the companion app and the IoT device. This can be achieved in several ways including: (1) Attacker getting within the communication range of the victim device with their own relay device. (2) Attacker taking control of the victim relay device that is in communication range of the IoT device.

For the first case, possible scenarios are hotels and Airbnb [49] that have IoT devices installed for their customers. The customers who have access to the physical place (hotel/Airbnb) can intentionally modify the firmware of the devices running in that place by exploiting their DFU mechanisms.

For the second case, the attacker can install a malicious app, which the attacker controls, on the victim's relay device. The attacker can program the app in way that the malicious firmware is transferred when DFU is performed. Achieving malicious activity on the victim's relay device through malicious apps is a common attack vector in mobile devices [14], [55], [57].

Essentially, the above scenarios allow the attacker to modify the communication with the target IoT device. By doing so, the attacker can modify the firmware running on the IoT device by abusing the IoT DFU mechanism.

In our threat model, we exclude the attacks that require the attacker to have wired physical access to the SoC (System on Chip) of the IoT Device.

3.2. IoT DFU Vulnerabilities

Our goal in this paper is to perform a large-scale analysis of vulnerabilities affecting IoT DFUs. To this aim, we first categorize these attacks into the following three categories.

Firmware Downgrade Attack (DownAttack) This attack reverts the firmware of the target IoT device to an older version. Reverting to an older firmware image may allow an attacker to re-introduce, and then exploit, vulnerabilities in older firmware or disable security features present in the updated version.

Device Bricking Attack (BrickAttack) This attack renders the target IoT device unusable. An attacker capable of performing the Device Bricking can launch this attack to perform DoS (Denial of Service) causing monetary damage to victim users or use the Device Bricking as an intermediary step to perform a more complex malicious activity, e.g., disabling security cameras, trackers, and motion sensors, to evade detection.

Firmware Modification Attack (ModAttack) This attack results in arbitrary execution of modified firmware on the target IoT device. Being able to arbitrarily modify the firmware images allows an attacker to run arbitrary code on the IoT device. An attacker who can perform the ModAttack has capabilities ranging from compromising privacy to harming the victim user. An attacker able to arbitrarily modify the firmware can, consequently, also perform a Device Bricking Attack (by substituting the original code with non-working code) or a Firmware Downgrade Attack (by replacing the original code with an older version).

To categorize the IoT devices based on their vulnerability to these attacks, we formulate an analysis called *Attack-Tester*, which is described in the following section.

3.3. Attack-Tester Analysis

Attack-Tester is a manual analysis for testing IoT devices and determining if they are affected by the studied categories of DFU vulnerability. Here, we explain the analysis process of *Attack-Tester* (Figure 3).

The *Attack-Tester* analysis requires knowledge of the IoT DFU mechanism (knowledge required to trigger the DFU and transfer firmware binary) and firmware binaries of the IoT Device as inputs to perform the testing and categorization. First, we reverse engineer the IoT DFU mechanisms by analyzing the interaction of the IoT devices and their companion apps, and we obtain the firmware binaries using our *AoT-Scout* analysis, described in Section 5.3. After our *AoT-Scout* analysis, we modify the original updated firmware binary by changing a few bytes and transferring it to the IoT device. We assign the aforementioned attack categories based on the state of the

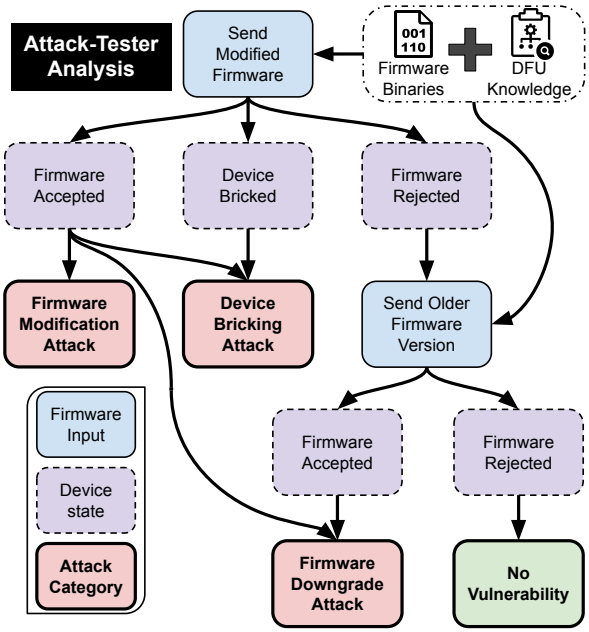

Figure 3: Overview of the *Attack-Tester* DFU vulnerability categorization analysis.

IoT device after the transfer of the modified firmware binary. Specifically, after transferring the modified firmware binary, we can observe the device in one of the following three states:

1) The IoT device accepts and executes the modified firmware.
2) The IoT device rejects the modified firmware and reverts to the previous firmware.
3) The IoT device becomes unresponsive or unusable.

An IoT device is vulnerable to **Firmware Modification Attack** if it accepts and executes the modified firmware. We verify the modified firmware is transferred and executed by checking the bytes transferred to the IoT device and the firmware version number of the firmware. If the transferred bytes and firmware version match the modified bytes and version of the modified firmware respectively, then the IoT device is vulnerable to Firmware Modification. In certain cases, to make the IoT device accept the modified firmware, we need to provide valid checksums or signatures and 're-sign' the binary. The details of 're-signing' are discussed in Section 7.4. We modify the firmware by changing only one or two bytes that are part of a string in the firmware binary. Modifying the firmware binary bytes in the code section without sufficient knowledge about their functionality can result unforeseen changes in the code execution which might make firmware binary unusable. More modifications can be done by understanding the firmware binary to achieve the required goal without making the device faulty. However, understanding the firmware binary comprehensively is out of scope of this paper.

IoT devices that are bricked by the modified firmware are vulnerable to the **Device Bricking Attack**. We verify the IoT device is bricked by checking if the IoT device is unresponsive or unusable. If the IoT device is unresponsive or unusable, then the IoT device is vulnerable to Device Bricking.

If the IoT device rejects the firmware, it means there is some mechanism in place for detecting firmware modifications which we are not able to bypass. For the devices that reject the firmware, we try to send older firmware versions that are retrieved during the *AoT-Scout* analysis described in Section 5.3. If the IoT device accepts the older firmware, then the IoT device is vulnerable to the **Firmware Downgrade Attack**.

As explained in Section 3.2, IoT devices vulnerable to ModAttack are also considered vulnerable to Device Bricking Attack (DBA) and Firmware Downgrade Attacks (FDA). In fact, BrickAttack can be performed by sending a modified firmware that bricks the victim IoT device. Likewise, DownAttack can be performed by replacing the firmware image with an older version. Consequently, if a device accepts an arbitrarily modified firmware binary, it is vulnerable to all three attacks.

By using the *Attack-Tester* categorization, we can effectively test IoT devices and identify their corresponding firmware update vulnerabilities. This analysis allows us to simulate various attack scenarios and test the device's resilience to these attacks, thereby providing valuable insights into potential vulnerabilities and weaknesses.

4. Methodology

In this paper, we study the security of firmware update mechanisms of IoT devices, and our study has two main phases. First, we focus on IoT DFU security reconnaissance (Section 5). Second, we perform a large-scale analysis of IoT companion apps (Section 6).

In the first phase, we manually analyze the security of the DFU mechanisms of IoT devices and use the results to perform, in the second phase, an automated large-scale analysis. This is shown in Figure 1, where the *AoT-Scout* and *Attack-Tester* analyses represent the first manual phase and their results are used, in the second phase, as inputs for the automated *App-Rumbling* pipeline.

In the first phase (Section 5), our goal is to determine if popular devices or apps fall under any vulnerability categories discussed in Section 3.2 and to generate fingerprints of vulnerable SDKs used by the affected companion apps. To do so, we collect Bestseller IoT devices from Amazon and sample top-ranked companion apps from the IoTSpotter dataset [42], and we perform a security analysis of the DFU mechanisms using our *AoT-Scout* analysis (Section 5.3). We refer to this phase as *Reconnaissance*.

In the second phase (Section 6), we perform a large-scale analysis of the companion apps on the Android marketplaces using the categorized fingerprints obtained in the first phase. For the second phase, our goal is to estimate the number of vulnerable companion apps on the Android marketplaces and the number of vulnerable devices that are affected by these apps. We refer to this phase as *App-Rumbling*.

5. IoT DFU Security Reconnaissance

In this phase, our goal is to generate fingerprints of the companion app SDKs that are affected by firmware update vulnerabilities. To achieve this goal, we need to obtain IoT devices and their companion apps so we can analyze the security of their DFU mechanisms using our *AoT-Scout* Analysis (5.3). To get representative and impactful data for our analysis we select two sources of data: (1) Bestseller IoT devices from Amazon (Section 5.1) and (2) Top-ranked companion apps from the IoTSpotter dataset [42] (Section 5.2). We refer to the dataset of devices, and their corresponding companion apps, obtained from these sources as ***DeviceDataset***.

We choose two sources of data because each source has its own advantages and disadvantages. The Amazon Bestseller list ranks the products based on the number of sales. Consequently, for the Amazon data, we cover the SDKs of the IoT devices that are popular among the consumers. The IoTSpotter dataset comprises companion apps from the Google PlayStore. The IoTSpotter dataset ranks SDKs by the number of companion apps that use them. We sample apps that are using the top-ranked SDKs from the IoTSpotter dataset. Consequently, for the IoTSpotter data, we cover the SDKs that are popular among the developers.

On the contrary, if we only targeted the top-ranked SDKs from the IoTSpotter dataset, we would miss the proprietary SDKs that are not open sourced and are not included in the IoTSpotter dataset. Although few apps use proprietary SDKs, a substantial number of consumers use their IoT devices. Similarly, if we only targeted the popular IoT devices from Amazon, we would miss the SDKs that are used by many companion apps even though the devices controlled by those companion apps are comparatively less popular among the consumers.

In the following sections we discuss the details of our *DeviceDataset* collection and our *AoT-Scout* analysis.

5.1. Bestseller IoT Devices Collection

To perform DFU security analysis of popular and widely used IoT devices, we consult specific categories of Bestseller lists on Amazon. The rank in categorical Bestseller lists on Amazon is an indicator of how much a product is selling in a particular category [8]. In particular, we focus on the 16 Bestseller categories that are likely to contain IoT devices (listed in Appendix B in Table 2). Since these lists are updated hourly, we captured a snapshot of them on April 1st, 2022, by using web scraping.

Sampling Bestseller Products. Each Bestseller list has 100 products ranked in the order of their sales. Since the total number of products from all the Bestseller lists amounts to more than 1,000 products, it is not feasible to analyze all of them. While we need to have a feasible number of IoT devices for analysis, our goal is to cover a wide variety of devices. To create a reasonably sized dataset and cover a variety of devices, we restrict our sampling to the top 6 devices from each Bestseller category and remove the devices that do not satisfy our definition of an IoT device (as detailed in Section 2).

Filtering IoT Devices. After removing the non-IoT devices, we are left with 41 IoT devices. To ease our analysis, we formulate criteria that restrict our study to specific IoT devices. Specifically, from the 41 IoT devices we remove the devices that: (1) Cost more than $50. (2) Require additional devices to operate and cannot function on their own. After removing the devices that do not satisfy our criteria, we are left with 30 IoT devices. This process helps us focus on those devices that are affordable

and used by a wide range of consumers. However, this filtering process also introduces certain limitations and biases that we discuss in Section 8.

Identifying Companion Apps. We identify the companion apps of the selected IoT devices by using the information given on the product page on Amazon and we compile a list of the companion apps' package names. We identify 19 unique companion apps.

Out of the 19 identified companion apps, 8 companion apps control multiple devices. Since we are interested in studying the companion apps' SDKs' DFU mechanisms, we want to avoid buying multiple devices communicating with the same companion app. To this aim, we determine, for each device the package name of its companion app and select the device that has the highest ranking among the other devices that are controlled by the same app.

In the end, we are left with 19 IoT devices. We download the companion apps of the 19 IoT devices from the Google PlayStore, and we add them to our *DeviceDataset*. We use these 19 apps/devices as input to our *AoT-Scout* analysis (Section 5.3) to analyze their DFU mechanism.

5.2. Top-Ranked Companion Apps Collection

In addition to what is explained in Section 5.1, we also use the IoTSpotter dataset [42] to sample companion apps for our analysis. Specifically, we consult the IoTSpotter dataset to cover the SDKs that are popular among the developers and to obtain the companion apps that use those SDKs.

IoTSpotter Dataset. IoTSpotter analyzed 2,182,654 apps on the Google PlayStore and identified apps that are IoT companion apps using a machine learning classifier. IoTSpotter classifies 37,783 apps as IoT companion apps. IoTSpotter identifies the 'IoT-related' class package names in the companion apps by using the number of companion apps that use the SDKs as a heuristic. IoTSpotter clusters IoT-related class package names based on similar prefixes. IoTSpotter ranks the clusters by using the number of companion apps that use the SDKs as a metric. IoTSpotter manually analyzes the top 50 clusters and identifies 11 SDKs that are present in the greatest number of companion apps. We manually analyze the 11 SDKs and only consider those that are involved in IoT DFU. We will refer to these SDKs as *IoTSpotter-SDKs*.

Sampling Companion Apps. To sample the companion apps that use the *IoTSpotter-SDKs* we use the apps' number of downloads as a ranking metric. For each *IoTSpotter-SDK*, we scrape the number of downloads of the companion apps that use that SDK from the Google PlayStore. We group the companion apps based on their number of downloads. Starting from the group with the highest number of downloads, we randomly select an app and manually check if the app meets the following criteria:

1) We confirm if the app is an IoT companion app by manually checking the Google PlayStore page of the app.
2) We check whether the companion app is involved in DFU by decompiling the app and manually analyzing the decompiled code of the app.
3) We check whether we can acquire an IoT device that the companion app controls.

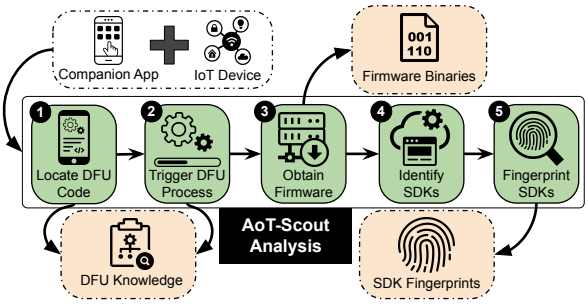

Figure 4: AoT-Scout Analysis.

4) We verify if the IoT Device fulfills the device selection criteria mentioned in Section 5.1.

When we find a companion app that satisfies all the above criteria, we include the app in our sample and repeat the process for the next *IoTSpotter-SDK*. If we have already analyzed an SDK from a companion app in Amazon Bestsellers, we skip the SDK. We sample one app for each remaining SDK and acquire its corresponding IoT device from Amazon. In the end, we are left with 1 IoT device and its companion app, and we add it to our *DeviceDataset*. We use the device (and its companion app) as input to our *AoT-Scout* analysis (Section 5.3) in order to analyze its DFU mechanism.

5.3. AoT-Scout Analysis

In this section, we describe our *AoT-Scout* manual analysis. The goal of this analysis is to study the DFU mechanisms of the companion apps and their IoT devices. To this aim, the *AoT-Scout* analysis: (1) generates fingerprints of companion app SDKs that are involved in DFU mechanisms; (2) obtains firmware binaries and knowledge about the DFU mechanisms for our *Attack-Tester* analysis (Section 3.3).

In turn, to achieve these goals, we systematically reverse engineer the DFU mechanisms of IoT Devices by following these steps:

1. Locate app DFU code;
2. Trigger DFU process;
3. Obtain firmware binaries;
4. Identify SDKs;
5. Fingerprint SDKs.

These steps are illustrated in Figure 4 and explained in detail in the following sections.

5.3.1. Locate app DFU code. In this step, we have companion apps and IoT devices as inputs. We want to locate the code involved in DFU in the companion apps. We start the analysis by decompiling the companion apps to get their Java source code and resource files. We use jadx [3] to decompile the apps. We check the resource files to see if any firmware binary is packaged within the companion app. If a binary is found, we locate if and where the binary is being accessed in the app code. To achieve this goal, we perform string search on the Java code using the firmware file name and location as the search keyword. We also perform string search on the Java code to locate code potentially involved in the DFU and

APIs involved in communication with other devices. For example, Android Bluetooth and network APIs. We use search keywords including 'firmware', 'DFU', 'update', and method names of the communication APIs.

After locating the aforementioned code, we dynamically analyze that code and the network communication of the companion app using a physical Android device. To perform dynamic analysis, we first achieve superuser privileges on the Android device we are using for the analysis. We achieve superuser privileges by 'rooting' [32] the Android device using an Android rooting app called Magisk [81]. By having superuser privileges, we can use the objection framework [64] and the HttpCanary [2] app to bypass SSL Certificate pinning. Bypassing SSL Certificate pinning allows us to analyze companion apps' network traffic. We use the HttpCanary app for analyzing the network traffic. To sniff the Bluetooth traffic, we enable Bluetooth HCI Snoop log in the Android device settings which enables us to capture Bluetooth traffic using WireShark [68]. For dynamically analyzing code, we use Frida [54]. Frida allows us to hook Android app methods for performing dynamic instrumentation. By using Frida, we can alter the companion app's method implementations, print logs, analyze stack traces, and study how the companion app communicates with its IoT Device and backend server.

We monitor the network communication of the companion app and check if any information about the firmware, such as firmware version or firmware binary URL, is being sent to or from the IoT Device and the backend server. If we detect such firmware information, we analyze decompiled app code to locate where the firmware information is handled, requested, received, and parsed. We locate the app code handling such firmware information because processing the firmware information is an essential part of the DFU mechanism, and our goal is to locate the code involved in the DFU mechanism.

In most cases, the firmware information is also displayed in the app UI. We analyze the app UI to locate the code that generates the UI displaying the firmware information. We locate the relevant UI code by searching the location of displayed strings and images in the app code. Then, we trace back the UI code to the code location where firmware information is handled.

At the end of this step, we have located the companion app code involved in the DFU of the controlled device.

5.3.2. Triggering the DFU mechanism. After pinpointing the code that handles the DFU, we check if we can trigger the DFU mechanism. To achieve this goal, we exploit the fact that most companion apps compare the current firmware version of the IoT Device and the firmware version available in the *firmware repository* to trigger the DFU process. Recall, from Section 2.3, that by *firmware repository* we mean the location that hosts the firmware binaries. This can be the companion app itself or the backend server. The firmware repository is the companion app itself if the firmware binary is packaged within the app otherwise firmware repository is a backend server, and the firmware binary is downloaded from the backend server. The DFU process is triggered if a newer firmware version is available in the firmware repository than the current firmware version of the IoT device.

In case the IoT device is already on the latest firmware version then the DFU process is not triggered. In this scenario, we spoof the current firmware version by modifying the firmware version number to a lower one, so that the firmware repository believes that it needs to send a newer version of the firmware to the IoT device. We force-trigger the DFU process by spoofing the current firmware version that is advertised by the IoT Device to the Android app. We spoof version numbers by modifying app code statically and/or injecting code dynamically into the app. For static modification, we use Apktool [1] to disassemble the app and repackage it after modification. For dynamic code injection, we use Frida.

At the end of this step, we can successfully force-trigger the DFU mechanism of the IoT Device.

5.3.3. Obtaining firmware binary. After successfully triggering the DFU, the goal of the next step is to obtain the firmware binary of the IoT device. As explained in Section 2.3, triggering the DFU causes the companion app to retrieve the firmware in one of the following two ways:

1) The companion app retrieves the firmware binary packaged within the companion app's APK.
2) The companion app requests the URL of the firmware repository from the backend server.

For the first scenario, we locate the directory where the firmware is stored using our knowledge about the DFU process from the first step. For the second scenario, we intercept the network traffic to the backend server to obtain the packet containing the URL of firmware repository. We retrieve the firmware binaries by using the firmware repository URL. In some cases, we are also able to access the older versions of firmware by modifying the firmware version string in the URL (e.g., changing `http://firm.repo/ver_3.bin` into `http://firm.repo/ver_2.bin`). The older versions of the firmware can be useful for performing DownAttack in the *Attack-Tester* analysis.

At the end of this step, we have retrieved the current version of the firmware binary of the IoT Device and for some cases also the older versions of the firmware.

5.3.4. Identifying SDKs. In this step, our goal is to identify whether the DFU process is handled within an SDK in the companion apps of the IoT devices that are categorized as vulnerable by the *Attack-Tester* analysis. We need to identify SDKs in the companion apps, so we can generate their fingerprints in the next step. To identify the SDKs, we check if the class packages handling the DFU are not part of the main app packages and standard Android packages. We consider an identified SDK to be affected by the *Attack-Tester* vulnerability category.

At the end of this step, we have SDKs categorized into *Attack-Tester* vulnerability categories.

5.3.5. Fingerprinting SDKs. Our next goal is to fingerprint the identified vulnerable SDKs, so we can detect their presence in an app for our analysis in Section 6. For fingerprinting the SDKs, we formulate a regular expression that indicates the presence of that SDK in an app.

SafetyNot [36] has shown the usage of regular expressions for fingerprinting Android APIs. We use a similar approach to fingerprint the SDKs and formulate regular

expressions for each SDK. However, Android code obfuscation can interfere with this step of our analysis. Code obfuscation is used to prevent the reverse engineering of Android apps. As shown by Dong et al. [26], most Android apps use ProGuard [33] to obfuscate their code. ProGuard is the most used obfuscation tool because it is the default obfuscation tool for Android. ProGuard obfuscates app code by changing class package names and method names. While formulating the regular expression for SDKs, we exploit the fact that hardcoded strings present in the app code are not obfuscated by ProGuard and that, in apps on the Google PlayStore, hardcoded strings are typically not obfuscated (as experimentally shown by Dong et al.). These regular expressions include hardcoded strings specific to the SDK vendor and string UUIDs involved in communication with the IoT devices. For example, the fingerprint: `no./nordicsemi./android./dfu` is a prefix for Bluetooth communication involving DFU information for IoT devices using a particular SDK. We refer to these regular expressions as *SDK-Fingerprints*.

Since we can encounter false positives if the *SDK-Fingerprints* are present in the non-DFU app code, we only target expressions that are present in the DFU mechanism's code of the companion apps. Using this approach allows us to target specific functionalities of the SDKs.

IoTSpotter also detects the presence of SDKs in Android apps. Specifically, IoTSpotter [42] utilizes LibScout [13], [24] to search for SDKs in Android apps. LibScout is a tool for detecting third-party libraries in Android apps. LibScout requires compiled jar files of the third-party libraries in order to detect them. Since the compiled jar files of all the SDKs from Section 5.3 are not available, we cannot use LibScout for detecting the usage of all SDKs. Furthermore, LibScout's approach also suffers from false positives as discussed in IoTSpotter [42]. We evaluate our approach against LibScout in Section 7.3.

At the end of this step, we have obtained *SDK-Fingerprints* of the vulnerable companion app SDKs.

We apply our *AoT-Scout* and *Attack-Tester* analyses to the *DeviceDataset*. From each dataset, we obtain a list of *SDK-Fingerprints* and their corresponding vulnerability category. We refer to this output as *Recon-Results*.

6. Large-Scale Analysis (App-Rumbling)

In this section, we present our large-scale analysis (*App-Rumbling*) of Android apps on the Google PlayStore. Since we cannot analyze and buy hundreds of different IoT apps and their corresponding devices, we exploit the assumption that the companion apps and their devices using the same SDKs are affected by the same bugs regarding the DFU process. We will show that this assumption holds in Section 6.2.

Our goal is to estimate the number of apps using vulnerable companion app SDKs. To this aim, we utilize the *Recon-Results*[1] obtained from our Reconnaissance phase (Section 5.3) and the 37,783 apps as IoT companion apps from the IoTSpotter dataset to perform this analysis. Specifically, in the *App-Rumbling* analysis we perform the following two steps:

1. *Recon-Results* contain the *SDK-Fingerprints* and their corresponding vulnerability category.

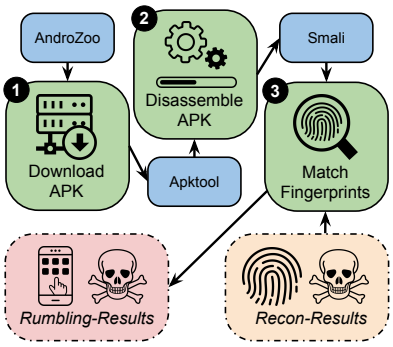

Figure 5: Overview of the large-scale app analysis using the *App-Rumbling* pipeline.

1) We use the *SDK-Fingerprints* to identify the vulnerable companion app SDKs in the companion apps from the IoTSpotter dataset (Section 6.1).
2) We use the *AoT-Scout* analysis to verify the presence of DFU vulnerabilities in the companion apps detected in the previous step (Section 6.2).

The details of these steps of the *App-Rumbling* analysis are explained in the following subsections.

6.1. SDK-Fingerprint Detection

This section explains the core steps of the *App-Rumbling* analysis that is assessing the prevalence of vulnerable companion app SDKs in the companion apps on the Google PlayStore. To perform this step, we utilize the *Recon-Results*[1] obtained from the previous section (Section 5.3). We analyze the 37,783 IoT companion apps from the IoTSpotter dataset using our automated *App-Rumbling* pipeline. Specifically for each app, we perform the following steps, which are also illustrated in Figure 5:

1) We download the application APK from AndroZoo [6].
2) We disassemble the APK into smali using Apktool [1].
3) We search the smali files using the *SDK-Fingerprints* obtained from the *Recon-Results*.
4) If we find matching *SDK-Fingerprints*, we assign the app the corresponding vulnerability category obtained from the *Recon-Results*.

As the output of this analysis, we obtain a list of companion apps and their corresponding vulnerability category. We will refer to this list as *Rumbling-Results*.

6.2. Vulnerability Verification

As the last step of our *App-Rumbling* analysis, we verify the presence of DFU vulnerabilities in the companion apps detected in the previous step (Section 6.1). To this aim, for each SDK in *Recon-Results*, we randomly sample one companion app from the *Rumbling-Results* that uses the SDK. For each sampled app, we acquire its corresponding IoT device using the criteria mentioned in Section 5.1. If we cannot find an IoT device that satisfies these criteria, we skip the app and sample another app. After acquiring the IoT devices, we perform the analysis of their DFU mechanisms using our *AoT-Scout* analysis and finally verify the presence of DFU

Device Name (Type)	App Name	SDK
Amazon Plug (Plug)	Amazon Alexa	Alexa (P)
ESICOO Smart Plug (Plug)	Cloud Intelligence	Aliyun [4]
Tracki 2022 (Tracker)	Tracki GPS	AltBeacon [7]
Daybetter LED (Light)	Apollo Lighting	Consmart (P)
Amazon Fire (Media Player)	Amazon Fire TV	Fling [9]
Govee H5075 (Env. Sensor)	Govee Home	Govee (P)
Hatch Mini (Speaker)	Hatch Sleep	HatchBaby (P)
Philips A19 (Light)	Philips Hue	Hue/Signify [52]/(P)
Tenmiro LED (Light)	KeepSmile	Inuker [25]
NIIMBOT D11 (Printer)	NIIMBOT	JcPrinter (P)
LandAirSea 54 (Tracker)	SilverCloud	LandAirSea (P)
ilumi LED A19 (Light)	New ilumi	Nordic [61]
SwitchBot Switch (Switch)	SwitchBot	NordicSecure [61]
NUT Key Finder (Tracker)	Findthing	NordicSecure [61]
Apple AirTag (Tracker)	Tracker Detect	*OBF* (P)
Blurams Dome (Camera)	blurams	*OBF* (P)
SYLVANIA LED (Light)	SYLVANIA	Telink [70]
Tile Mate 2022 (Tracker)	Tile	Tile (P)
Kasa KP115 (Plug)	Kasa Smart	Tplinkra (P)
Wyze v2 (Camera)	Wyze	Tutk [73]
Aoycocr Smart Plug (Plug)	Popotan	Tuya [38]
GoSund Power Strip (Switch)	Gosund	Tuya [38]
Wemo WSP080 (Plug)	Wemo	Wemo [77]

TABLE 1: Overview of the analyzed devices and apps. (P) = Proprietary SDKs. *OBF* = Obfuscated.

vulnerability using our *Attack-Tester* analysis. We include the IoT devices (and their companion apps) acquired in this step in our *DeviceDataset*.

7. Experimental Results

We evaluate and discuss the results of our study. First, we discuss the *Recon-Results* obtained from the IoT DFU security reconnaissance phase (Section 7.1). Then, we focus on the *Rumbling-Results* obtained from the large-scale *App-Rumbling* phase (Section 7.2).

The results from the first phase indicate the presence of vulnerable DFU mechanisms among the Bestseller IoT devices on Amazon and popular companion app SDKs. Specifically, there are 61 potentially vulnerable IoT devices among the top 50 Bestsellers from the 16 categories mentioned in Section 5.1 and 6 vulnerable companion app SDKs. The results from the second phase reveal that vulnerable SDKs are being widely used in the companion apps on the Google PlayStore. Specifically, there are 1,356 apps that are using at least one of the vulnerable companion app SDKs.

In summary, we analyzed 23 devices (with their corresponding companion apps) and identified 6 SDKs that are vulnerable to firmware attacks and 1,356 companion apps that are using these vulnerable SDKs. A summarized list of the 23 analyzed devices/apps devices is shown in Table 1 (Table 3 in Appendix B provides more details about these devices). We verified the firmware attacks using our *Attack-Tester* analysis on 8 devices from Amazon. Furthermore, we also identified 61 Bestseller IoT devices from Amazon IoT devices that are potentially vulnerable to firmware attacks.

7.1. Reconnaissance Results

In this section, we discuss the results of our DFU security analysis of *DeviceDataset*, which contains the IoT Devices and their corresponding companion apps gathered from Amazon and the IoTSpotter dataset, respectively.

Data Collection For compiling the data from Amazon Bestsellers, we sampled the 19 IoT devices and their 19 corresponding companion apps. These 19 devices were chosen after filtering out devices that did not satisfy our selection criteria. Filtered-out devices include non-IoT devices and non-bare metal devices like smartwatches and voice assistants. We also filtered out devices that did not communicate with companion apps such as cameras, speakers, and lighting products.

To obtain data from IoTSpotter, we analyzed the companion apps that were using the 11 top-ranked SDKs as described in Section 5.2. We manually investigated each SDK and found 3 SDKs that were involved in DFU mechanisms (*IoTSpotter-SDKs*). The remaining 9 SDKs were not involved in the DFU mechanisms. For the *IoTSpotter-SDKs*, we sampled companion apps that were using the SDK and selected one app for which we can acquire its corresponding IoT device that meets our selection criteria as described in Section 5.2. After removing duplicate SDKs found in Amazon Bestsellers' companion apps, we sampled 1 companion app and its IoT device.

***AoT-Scout* Analysis** We analyzed the devices/apps from *DeviceDataset* using our *AoT-Scout* and *Attack-Tester* analyses to obtain the *Recon-Results*. *Recon-Results* contains the list of companion app SDKs and their corresponding vulnerability category. We found 6 companion app SDKs that are vulnerable to firmware attacks. Specifically, the 6 SDKs are *Nordic-SDK* [61], *Tuya-SDK* [38], *JcPrinter-SDK*, *NordicSecure-SDK*, *Telink-SDK* [70], and *Wemo-SDK* [77]. The specific details of the SDKs are discussed as case studies in Section 7.4 and additional case studies are discussed in Appendix A.

***Attack-Tester* Analysis** We tested the 6 SDKs using our *Attack-Tester* analysis (Section 3.3) to categorize SDKs by their DFU vulnerabilities. We identified that 3 SDKs are vulnerable to **ModAttack** (Firmware Modification Attack): *Nordic-SDK*, *Tuya-SDK*, and *JcPrinter-SDK*. The IoT devices on which we can perform ModAttack are a smart switch, a smart plug, and a label printer for the *Nordic-SDK*, *Tuya-SDK*, and *JcPrinter-SDK*, respectively. As a consequence of ModAttack, these 3 SDKs are also vulnerable to DownAttack and BrickAttack. Additionally, we identified 1 SDK (*Telink-SDK*) that is exclusively vulnerable to **DownAttack** by testing DownAttack on a smart bulb. Finally, we identified 2 SDKs (*Wemo-SDK* and *NordicSecure-SDK*) that are exclusively vulnerable to **BrickAttack** by testing BrickAttack on a smart plug and a smart tracker.

Throughout our attack tests, we focus on manipulating companion apps as a way of demonstrating the potential DFU vulnerabilities of IoT devices. However, it is important to note that these attacks could also be achieved through communication manipulation. Specifically, by manipulating the communication protocols that are used to connect IoT devices, an attacker can potentially send malicious firmware directly to the IoT device.

We gained interesting insights after our analysis of the *DeviceDataset*. Most IoT devices utilizing proprietary SDKs use obfuscation techniques to prevent reverse engineering and security mechanisms in place to prevent reverse engineering. On the contrary, most IoT devices utilizing open-source SDKs do not utilize such comprehensive defensive mechanisms in place that can prevent

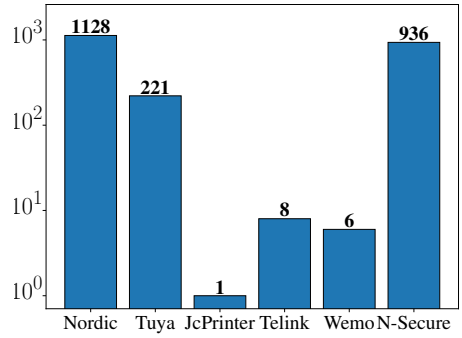

Figure 6: Logarithmic-scaled distribution of *Rumbling-Results* showing the usage of potentially vulnerable SDKs in the 1,356 companion apps from Google PlayStore. (N-Secure is the *NordicSecure-SDK*.)

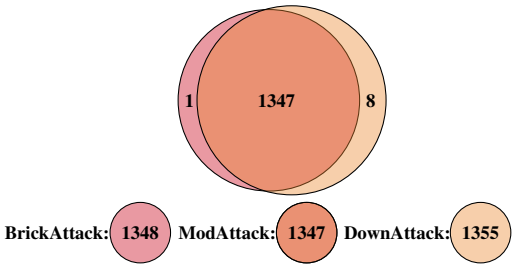

Figure 7: Venn diagram (not in scale) showing the number of companion apps that use vulnerable SDKs for **BrickAttack**, **ModAttack**, and **DownAttack**.

firmware attacks. However, as shown by our results the proprietary SDKs are vulnerable to firmware attacks. We also found that the IoTSpotter data covers mostly open-sourced SDKs, but it misses many proprietary SDKs used by the popular devices in the Amazon Bestseller lists.

To summarize, in the Reconnaissance phase of our study, we manually analyzed 20 devices (with their companion apps) from the *DeviceDataset*, and we identified 6 vulnerable SDKs. We fingerprinted each of the vulnerable SDKs and organized them into a comprehensive list that we refer to as *Recon-Results*. The additional 3 devices in the *DeviceDataset* will be used for verification (see Section 6.2).

7.2. App-Rumbling Results

This section discusses the results of the large-scale analysis of apps on the Google PlayStore using our *App-Rumbling* pipeline from Section 6. We used *Recon-Results* as the input to the *App-Rumbling* pipeline to search for vulnerable SDK fingerprints in the 37,783 companion apps list from the IoTSpotter dataset to obtain *Rumbling-Results*. *Rumbling-Results* comprise the 6 potentially vulnerable SDKs and the companion apps that use those SDKs. We identified 1,356 companion apps on the Google PlayStore that use at least one of the 6 vulnerable SDKs. The *Rumbling-Results* are shown in Figure 6.

SDK-Fingerprint Detection As can be seen in Figure 6, *Nordic-SDK* and *Tuya-SDK* are used widely among the companion apps. *NordicSecure-SDK* is also used by a significant number of companion apps. The reason for their wide usage is that these three SDKs are open-source and developed by the vendors of the SoCs (System on Chip) used in IoT devices. The IoT device vendors prefer to use the SDKs developed by the SoC vendors because it is feasible to integrate the SDKs with their IoT devices and companion apps. *JcPrinter-SDK* is the only proprietary SDK that is vulnerable to firmware modification attacks. Since it is a proprietary SDK, it is not available to the public and hence it is not used by many companion apps. However, *JcPrinter-SDK* still has a high impact since it is used by popular Bestseller IoT Devices.

Wemo-SDK is vulnerable to device bricking attack. *Wemo-SDK*'s DFU mechanism involves sending signed and encrypted firmware to the IoT devices. However, if a firmware image with an invalid signature is sent, this results in a device bricking attack, as the IoT devices using the *Wemo-SDK* do not present a recovery strategy. *Wemo-SDK* was open source in the past, however, the developer decommissioned [39] the SDK, which explains its low usage. Nevertheless, *Wemo-SDK* still has a high impact as it is used by Bestseller IoT Devices.

Telink-SDK is vulnerable to firmware downgrade attack. *Telink-SDK* is open source, and the company is also an SoC provider, however, *Tuya-SDK* provides wrappers around the *Telink-SDK*. Since *Tuya-SDK* is widely used, this leads to lower usage detection of only *Telink-SDK*. We formulate our *SDK-Fingerprints* to detect *Telink-SDK* only if the *Tuya-SDK* is not used in the app.

Note that the sum of apps in Figure 6 is greater than 1,356 because some apps use multiple SDKs. Specifically, we found 933 apps that use more than one SDK. Apps using more than one SDK are designed to control different IoT devices from different vendors.

Impact of *Rumbling-Results* We show the distribution of firmware attacks in the companion apps from the *Rumbling-Results* in Figure 7. There are 1,355 apps vulnerable to DownAttack, 1,347 apps vulnerable to ModAttack, and 1,348 apps vulnerable to BrickAttack. As discussed in Section 3.3, vulnerabilities to ModAttack also result in vulnerabilities to DownAttack and BrickAttack. As shown in Figure 7, most DownAttacks and BrickAttacks are caused by the fact that most companion apps are vulnerable to ModAttack.

From the numbers in Figure 7, we excluded 3 apps that are using only *NordicSecure-SDK*. In fact, during the responsible disclosure process (see Section 7.5), by communicating with the Nordic Semiconductor company, we discovered that, while the devices in our dataset using *NordicSecure-SDK* are vulnerable, it is in theory possible to use this SDK safely. We provide more details about this SDK in Section 7.4.

To extend our evaluation of the impact of *Rumbling-Results*, we identify which companion apps control the top 50 Amazon Bestsellers devices from the 16 categories mentioned in Section 5.1. We found 24 companion apps from the *Rumbling-Results* that control 61 potentially vulnerable IoT devices among the top 50 Amazon Bestsellers. These IoT devices include smart lighting, speakers, doorbells, cameras, environment sensors, and trackers.

We further extend the scope of our evaluation by gathering the number of downloads of apps in the *Rumbling-Results* from Google PlayStore. Figure 8 shows the distribution of the number of downloads of the affected apps. The apps in the top download categories control IoT

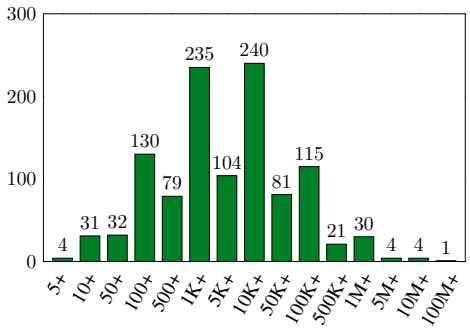

Figure 8: Distribution of number of **app downloads** of the 1111 companion apps from the *Rumbling-Results* that are potentially affected by firmware update vulnerabilities. The chart shows downloads of 1111 apps since for 245 apps download information was not available.

devices such as smart home products and fitness bands.

Vulnerability Verification As mentioned in Section 6.2, for each vulnerable SDK, we verified the vulnerability by performing the firmware attack on one IoT device and its companion app sampled from the *Rumbling-Results*. We verified the vulnerabilities for *Nordic-SDK*, *NordicSecure-SDK*, and *Tuya-SDK* on one additional IoT device and its companion app. Hence, we performed vulnerability verification on 3 devices (and their apps). For *JcPrinter-SDK*, *Wemo-SDK*, and *Telink-SDK*, we were unable to acquire an additional IoT device that satisfied our selection criteria since these SDKs are not widely used.

7.3. Results Validation

We compared our *SDK-Fingerprints* detection against the result of LibScout as used by IoTSpotter for each SDK present in both the IoTSpotter dataset and our *App-Rumbling* results. Three SDKs that are present in both datasets: *Nordic-SDK*, *Tuya-SDK*, and *Telink-SDK*. For each of these SDKs, we took the difference between the set of apps detected by our *App-Rumbling* and the set of apps detected by LibScout in the IoTSpotter dataset. Specifically, we were looking for apps that were detected by our *App-Rumbling* analysis pipeline but not by LibScout. We will refer to these apps as *Evaluation-AppSet*. Essentially, the apps in *Evaluation-AppSet* are either false negatives of LibScout or true positives of our *App-Rumbling* analysis. We manually analyzed the *Evaluation-AppSet* and confirmed that the apps were indeed using the SDKs. Specifically, our *App-Rumbling* pipeline detected 4 Tuya-SDK and 6 Telink-SDK apps that were not detected by LibScout. For *Nordic-SDK*, we sampled 5 apps out of 600 apps that were detected by our *App-Rumbling* pipeline but not detected by LibScout. In summary, our evaluation shows a 100% true positive rate for all the apps in *Evaluation-AppSet*, demonstrating that our approach is better suited for the detection of libraries handling DFU than the LibScout approach.

7.4. Case Studies

In this section, we discuss the details of the DFU mechanisms of four identified vulnerable SDKs. We provide an additional case study in Appendix A.

Nordic and Nordic-Secure. In this case study, we discuss the *Nordic-SDK* [61] that we identify during our analysis of *DeviceDataset* in Section 5. *Nordic-SDK* is developed by Nordic Semiconductor [60] SoC vendor. During our analysis, we discovered two variants of the SDKs: one without firmware signature verification and one with firmware signature verification. We refer to the one without verification as *Nordic-SDK* and to the other as *NordicSecure-SDK*. Using our *Attack-Tester* analysis, we verified that *Nordic-SDK* is vulnerable to firmware modification attack since the SDK has no firmware signature verification. However, *NordicSecure-SDK* is only vulnerable if the SDK is misconfigured. We generated *SDK-Fingerprints* for both SDK variants. Furthermore, we ensure that each fingerprint only indicates the presence of its specific variant and does not trigger the detection of the other variant. After our *App-Rumbling* analysis, we detected *Nordic-SDK* in 1,128 apps and *NordicSecure-SDK* in 936 apps.

We identified the ***NordicSecure-SDK*** being used in the SwitchBot [79] companion app. The Bestseller device we analyzed is the 'SwitchBot Smart Switch Button Pusher' [66]. The SwitchBot Switch uses Nordic Semiconductor's nRF52 Series SoC. This SoC series supports communication via Bluetooth. After reverse engineering the companion app, we discovered that the firmware is packaged within the APK. When performing DFU, the firmware is sent to the SwitchBot Switch using Bluetooth. By manually analyzing the firmware, we found that the firmware is in compressed zip format, and it includes the main firmware binary, a .dat file, and a .json file containing metadata about the firmware.

To test the firmware modification attack, we modified the firmware binary that is packaged in the APK and transferred it to the SwitchBot Switch. The SwitchBot Switch rejected the modified firmware. However, after extensively analyzing the app and researching the SDK firmware formats, we discovered that specific bytes in the .dat file were the CRC of the firmware. We reverse engineered the CRC algorithm by analyzing firmware packaging tools from Nordic Semiconductor, generated the new CRC for the modified firmware, and packaged the modified firmware using Nordic Semiconductor's firmware packaging tool. The SwitchBot Switch accepted the modified firmware with the updated CRC. We confirmed the transfer of the modified firmware by ensuring the firmware version that the SwitchBot Switch is running after the DFU is the same as the firmware version of the modified firmware. The current firmware version was confirmed using the firmware version displayed in the SwitchBot App and the firmware version communicated by the SwitchBot Switch via Bluetooth.

For verification of *NordicSecure-SDK* vulnerability, as mentioned in Section 6.2, we sampled the Findthing companion app. We analyzed and tested the DFU mechanism of the Findthing app and its IoT device and realized that the Findthing app uses firmware signature verification. In this case, the signature is included in the .dat file in the compressed zip file. We tested the tracking device for firmware modification attack by modifying the firmware binary and sending it to the IoT device. Our modified binary made the tracking device inoperable (bricked). Specifically, it stopped interacting with the companion app and did not respond to any hardware reset commands

using the buttons on the tracking device. We confirmed this device bricking attack by performing the attack on two additional similar tracking devices. We were not able to perform the firmware downgrade attack on this device because we were not able to retrieve signed firmware having a lower version number.

For ***Nordic-SDK***, without the signature verification, the IoT device is vulnerable to ModAttack. We confirmed the ModAttack on the 'ilumi smart bulb' using the 'New ilumi' companion app [37]. It should be noted that *Nordic-SDK* is vulnerable to ModAttack by design and there is no defensive mechanism implemented in *Nordic-SDK* that can prevent this attack.

As part of our Coordinated Vulnerability Disclosure, we reported this issue to Nordic Semiconductor. We received a development kit [63] from Nordic Semiconductor for testing and verifying our findings. For *NordicSecure-SDK*, we discovered that it is vulnerable to device bricking attacks and firmware modification attacks only if the SDK is misconfigured. Specifically, if the Nordic Semiconductor's firmware packaging tool is misconfigured, it can package firmware without a signature, which makes the firmware vulnerable to modification attacks. This misconfiguration is what makes, for instance, the SwitchBot Switch vulnerable to ModAttack.

Regarding device bricking attacks, we discovered that they are possible when the IoT device receives incorrectly signed firmware and there is no mechanism to revert to the old firmware. Specifically, when the IoT device detects an incorrect firmware signature, it discards the incorrectly signed firmware and waits for the companion app to send a correctly signed firmware. While in this 'waiting' state, the IoT device cannot be operated normally by the end user. If no correctly signed firmware is sent to the IoT device, it will remain inoperable (bricked) indefinitely. This issue happens, for instance, in the Findthing tracking device in our dataset. Specifically, after we send the incorrectly signed firmware to the tracking device, it discards the firmware and enters the 'waiting' state.

After further discussion with Nordic Semiconductor, we discovered that there are two mechanisms to recover from the 'waiting' state. The first mechanism is to send the correctly signed firmware to the tracking device using the companion app. However, this mechanism is not available to the end user in the Findthing tracker's companion app. The second mechanism is only available if the IoT device supports dual-bank firmware updates [62]. Essentially, dual-bank firmware updates allow the IoT device to have two firmware images. When the IoT device receives a firmware update, it first writes the firmware to the secondary bank for verification. If the firmware is correctly signed, the IoT device overwrites the primary bank firmware and runs the new firmware. If the firmware is incorrectly signed, the IoT device discards the firmware and continues to run the firmware from the primary bank. We confirmed this behavior of *NordicSecure-SDK* by performing the BrickAttack on the development kit that we received from Nordic Semiconductor. However, if the IoT device does not have enough storage to support dual-bank firmware updates, it will overwrite the primary bank firmware with the incorrectly signed firmware. In this case, the IoT device will remain inoperable (bricked) indefinitely until the companion app sends the correctly signed firmware. The tracker in our dataset does not support dual-bank firmware updates and does not have a mechanism to recover from the 'waiting' state so it is vulnerable to BrickAttack. However, the tracker can be recovered from the 'waiting' state by sending the correctly signed firmware using Nordic Semiconductor's 'nRF Connect for Mobile' app [11].

In summary, while, in theory, it is possible to use the *NordicSecure-SDK* in a way in which no attack against DFU is possible, all the devices in our dataset using *NordicSecure-SDK* use it incorrectly, and, for this reason, they are exposed to DFU attacks. On the other hand, *Nordic-SDK* is vulnerable to ModAttack in all configurations.

Tuya. Tuya [75] is a smart device vendor that provides both ready-to-use smart devices and an SDK to control the devices. These devices mostly communicate via Wi-Fi and use the Espressif Systems' ESP8266 SoC. We identified *Tuya-SDK* during the analysis of *DeviceDataset* in the Popotan companion app for the Aoycocr Smart Plug. We discovered a tool called tuya-convert [22] that is used to modify firmware of the devices using the *Tuya-SDK*. Tuya-convert relies on exploiting the MQTT broker as discussed in Section 2. When the IoT device is registering to the MQTT broker, tuya-convert impersonates as the broker and connects to the IoT device. After a successful connection, the tuya-convert's MQTT broker triggers the DFU and sends the modified firmware. Since the modified firmware is successfully flashed on the IoT device, the *Tuya-SDK* is vulnerable to ModAttack. We verified the firmware modification vulnerability on the GoSund companion app that controls a smart switch.

Telink. Telink [70] is an SoC vendor that provides SDKs for Android apps and is the developer of *Telink-SDK*. We identified *Telink-SDK* in the SYLVANIA Smart Home companion app from the *DeviceDataset*. We analyzed the DFU mechanism of the SYLVANIA app with the SYLVANIA SMART+ Bluetooth Soft White BR30 LED BULB. The SYLVANIA Bulb communicates with the SYLVANIA app using Bluetooth. The firmware binaries used by the SYLVANIA Bulb are encrypted and signed. We tested the firmware modification attack on the SYLVANIA Bulb, but the modified firmware was rejected by the device. In fact, the SYLVANIA Bulb reverted to the original firmware after signature verification of the firmware failed.

We retrieved the older versions of the firmware from the SYLVANIA app's backend server by tweaking the URL of the firmware update binary as mentioned in Section 5.3. The older firmware binaries we retrieved were already signed and encrypted by the backend server. We tested the firmware downgrade attack on the SYLVANIA Bulb using the older firmware binaries. We successfully downgraded the SYLVANIA Bulb to the older firmware version. We confirmed the downgrade attack by verifying the decrement in the version number of the firmware currently running on the SYLVANIA Bulb. While the *Telink-SDK* is vulnerable to the DownAttack, we found that it is not affected by ModAttack and BrickAttack. We were not able to find an additional device for verification of the DownAttack on *Telink-SDK*.

7.5. Coordinated Vulnerability Disclosure

Following the ethical guidelines of our community, we initiated the coordinated vulnerability disclosure process.

We contacted the vendors of the affected IoT devices to the best of our ability. At the time of this writing, we have contacted 9 vendors. We received acknowledgment of the presence of vulnerabilities from SwitchBot. As discussed in Section 7.4, we contacted *Nordic-SDK's* vendor (Nordic Semiconductor) and performed further investigation of the vulnerabilities on their development kit [63]. We discovered that *NordicSecure-SDK* is vulnerable to firmware attacks only if the SDK is misconfigured and *Nordic-SDK* is vulnerable to ModAttacks in all configurations. This investigation is discussed in detail in Section 7.4.

For *Wemo-SDK*, the vulnerability was triaged, by a crowdsourced security platform (BugCrowd [21]), as critical. However, *Wemo-SDK's* vendor (Belkin) decided to stop providing updates for the affected device since they have released a newer version of the affected device. We note that, however, the affected device was an Amazon Bestseller in April 2022, and, at the time of this writing (February 2023), it can still be purchased on Amazon.

8. Limitations and Future Work

The following sections discuss the limitations of the two main analyses performed in this paper.

Reconnaissance. Our first study (Section 5) involves the analysis of IoT devices using only Bluetooth or Wi-Fi. However, there are other ways for IoT devices to communicate with the companion app besides using Bluetooth or Wi-Fi. Two other popular methods of communication are by using ZigBee [5] or Z-Wave [86]. We do not include IoT devices using these communication methods in our study because they do not meet our device selection criteria (Section 5.1). As mentioned in Section 2.1, we exclude IoT Devices with certain characteristics from our study. Specifically, we exclude IoT devices that are not bare metal and IoT devices that only communicate with non-Android relay devices. Furthermore, we do not analyze devices that cost more than $50, as explained in Section 5.1. Limiting our analysis to the devices on the lower side of the cost spectrum could have potentially biased our results. While, in our investigation, we did not find any indication of this aspect, it is reasonable to assume that more expensive devices might be implemented following higher security standards. For this reason, a future direction is to evaluate whether the cost, as well as other factors (e.g., the targeted customers), are correlated with the overall security of an IoT device.

There are certain companion apps that we cannot reverse engineer properly. One reason for this issue is their usage of native code. Since our analysis methodology involves analyzing the Java code of apps, we consider apps relying on native code out of scope for this paper. This restriction further limits the range of IoT devices our analysis covers. Consequently, there are companion app SDKs that remain unevaluated by our analyses.

The DFU vulnerabilities we find by applying *Attack-Tester* are limited by our capability to manually reverse engineer the DFU mechanisms of the IoT devices using the *AoT-Scout* analysis. Although by using our methodology we can systematically study the DFU mechanisms, it is still possible that some of the devices that we are not able to attack have vulnerable DFU mechanisms. These false negatives can be eliminated by putting more effort into manual reverse engineering.

Large-Scale App Analysis. In our large-scale analysis, we try to the best of our ability to give a generalized picture of the IoT DFU mechanisms on the Google PlayStore. To the best of our ability, we try to formulate *SDK-Fingerprints* that counter obfuscation for detecting SDKs as mentioned in Section 5.3. Still, there can be apps that employ obfuscation techniques that are more sophisticated than the default Android ProGuard obfuscation, discussed in Section 5.3. The usage of such obfuscation techniques can cause *SDK-Fingerprints* to not be detected and result in false negatives in the *Rumbling-Results*. However, as previous studies [26] have shown, the usage of such obfuscation is almost non-existent on the Google PlayStore.

One cause of false positives in the *Rumbling-Results* is the presence of apps that use vulnerable SDKs but never actually execute that code. Our analysis only detects the presence of SDKs. However, as shown by our results in Section 7.2, the presence of an IoT SDK in an IoT companion app is a good indicator that the IoT SDK code is executed. Additionally, we do not analyze the bootloader of the IoT devices. For this reason, IoT devices that use a bootloader that is not part of the SDK to verify the updated firmware might lead to false positives. However, in our dataset, we have not encountered devices with this behavior. Another factor for false positives is when the companion apps implement their own security mechanisms, such as performing signature verification of the downloaded firmware, on top of the SDKs, or when they modify the SDKs in a way to secure IoT devices against the firmware attacks we identified. However, no such modifications are present in the apps that we manually analyzed, and such modifications would be overwritten every time an SDK is updated.

Future Work. Some steps of our analysis require manual effort. As future work, manual analysis can be automated. For example, the retrieval of firmware binaries can be automated by statically extracting URLs to the firmware repositories from the APKs and detecting and extracting binaries packaged within the APKs. We tried existing tools [53] for extracting URLs but they were not useful for our purpose. The tools failed to work on modern Android apps and cannot extract authentication tokens required to access URLs. The DFU code identification and *SDK-Fingerprints* generation can be automated by performing static analysis on the companion app code.

9. Related Works

IoT Device Firmware Updates. Prior works have investigated vulnerabilities in the DFU mechanisms of IoT devices. However, most of the prior works only focus on a single device or require physical access to the IoT device for flashing the firmware. Cui et al. [23] find vulnerabilities in the firmware update mechanisms of HP printers. They send modified firmware to the HP printers by sending specific commands to the printers to trigger the firmware update process. The methodology of Cui et al. differs from ours because they do not leverage companion apps for their attacks and only focus on HP printers.

Hernandez et al. [34] reverse engineered Google's Nest thermostat and found vulnerabilities in the firmware update verification. The attack of Hernandez et al. requires knowledge of the IoT device hardware and physical access

to the IoT device hardware via USB or UART port. Their attack intervenes in the boot process and injects modified firmware before the firmware verification is performed. Pen Test Partners [47] and Positive Technologies [69] investigate DFU vulnerabilities of IoT devices. Both works focus on specific IoT devices, while our approach aims to analyze IoT devices at a large scale. The authors conclude that having physical access to the IoT device can allow attackers to upload modified firmware. Their attacks differ from our approach because we do not require a physical connection to the hardware of the IoT devices. Andy et al. [10] discuss the possibility of injecting modified firmware into IoT devices that use the MQTT protocol. Their attack involves changing the URL of the firmware sent to the IoT Device. Ling et al. [46] perform a firmware modification attack on a smart plug using the device's Windows update tool.

IoT App Analysis. Prior works [18], [35], [48], [50], [59], [82] performed security analyses of IoT companion apps. IoTSpotter [42] analyzes the security of IoT companion apps by detecting the presence of CVEs in the SDKs used by the companion apps. Our study differs from IoTSpotter because none of the CVEs detected by IoTSpotter are related to DFU vulnerabilities or directly affect the security of the IoT devices. Furthermore, our study involves analyzing both the companion apps and the IoT devices.

Chatzoglou et al. [17] use existing tools and frameworks to analyze the security of IoT companion apps. Their study finds existing CWEs, third-party trackers and some other known issues in the companion apps. Casagrande et al. [15] reverse engineer communication protocols of Xiaomi fitness trackers by leveraging their companion apps. The authors find attacks involving man-in-the-middle, companion app impersonation, and eavesdropping on the communication between the companion app and the fitness trackers. Casagrande et al. also find methods to trigger the DFU but do not attempt to downgrade or inject modified firmware. Wang et al. [76] generate fingerprints of companion apps of vulnerable IoT devices. The authors utilize the fingerprints to find similar IoT apps and other potentially vulnerable IoT devices. This work differs from ours because the authors only investigate known CVEs that do not include the DFU attacks in our work. Neupane et al. [51] utilize existing tools to perform a security analysis of IoT apps. Neupane et al. find privacy policy violations and vulnerabilities such as TLS misconfigurations. Zhang et al. [87] analyze IoT apps for detecting vulnerabilities in the process of generating shared credentials for communication between the IoT device and the companion app. All these works do not focus on the DFU mechanisms of IoT devices, but explore other aspects related to apps' security.

Finally, other works study the security of Android apps' SDKs. Thomas et al. [72] explore the JavaScript-to-Java interface vulnerability in advertisement SDKs. Backes et al. [13] analyze Android apps to detect vulnerabilities caused by API misuse in third-party libraries. Derr et al. [24] detect vulnerabilities in Android apps due to outdated third-party libraries. Feal et al. [29] analyze the privacy violations caused by third-party libraries in Android apps. The methodologies of these works cannot directly be applied to our study as they do not consider the interplay between companion apps and the corresponding devices. On the contrary, we designed dedicated analyses to force the execution of the DFU mechanism.

IoT Device Firmware. Prior works investigated vulnerabilities in the firmware of IoT devices. Chapman [16] reverse engineers the firmware of a smart bulb to decrypt Wi-Fi credentials and inject network packets into the smart bulb. Stroetmann et al. [65] reverse engineer Kasa Smart plug's firmware and identify vulnerabilities including unauthenticated Wi-Fi commands sent to the device and improper TLS certificate verification. Wen et al. [78] present a static firmware analysis tool called FirmXRay, which identifies vulnerabilities in the Bluetooth pairing of IoT devices. Finally, other works [19], [56] analyze IoT apps to generate inputs for fuzzing the firmware of their corresponding IoT devices. In our study, we do not focus specifically on reverse engineering firmware. Conversely, we study companion apps to identify issues in the DFU mechanisms of IoT devices.

10. Conclusion

In this paper, we analyzed the security of the firmware update mechanisms adopted by IoT devices. Using our *AoT-Scout* and *Attack-Tester* analyses, we reverse engineered the firmware update mechanisms of Bestseller IoT devices and top-ranked companion apps. Specifically, we tested 23 companion apps and their corresponding devices against three categories of firmware update attacks.

Then, to scale our security analysis to thousands of apps, we identified specific SDKs used by companion apps to implement the firmware update process. Specifically, we found 6 SDKs that, when used to implement the DFU functionality, lead to insecure updates. For these SDKs, we first confirmed our findings on 8 real-world devices that adopt them. Then, we fingerprinted vulnerable SDKs and used our fingerprints to detect their presence on a large dataset of companion apps from the Google Play Store. Our results indicate the usage of vulnerable SDKs by 1,356 apps and 61 popular devices. Overall, our work highlights that a critical feature such as DFU is often not securely implemented by popular IoT devices and SDKs.

Acknowledgements

We are grateful to our reviewers for their valuable feedback. We are also grateful to companies Nordic Semiconductor and SwitchBot U.S. for providing support and working with us. We would also like to thank Everett Johnson for helping us with the manual analysis. This work has been supported by the INTERSECT project, Grant No. NWA 1160.18.301, funded by the Netherlands Organisation for Scientific Research (NWO), and by the Dutch Ministry of Economic Affairs and Climate Policy (EZK) through the AVR project 'FirmPatch'. This work was also supported in part by DARPA under contract number N6600120C4031, and by the Google's ASPIRE Award. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, the U.S. Government, or Google.

References

[1] Apktool. https://github.com/iBotPeaches/Apktool, 2022.

[2] Httpcanary. https://github.com/MegatronKing/HttpCanary, 2022.

[3] Jadx. https://github.com/skylot/jadx, 2022.

[4] Aliyun.com. Alibaba cloud iot. https://iot.aliyun.com/, 2023.

[5] Connectivity Standards Alliance. Zigbee. https://csa-iot.org/all-solutions/zigbee/, 2022.

[6] Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. Androzoo: Collecting millions of android apps for the research community. In *Proceedings of the International Conference on Mining Software Repositories (MSR)*, 2016.

[7] AltBeacon. Android beacon library. https://github.com/AltBeacon/android-beacon-library, 2023.

[8] Amazon. Amazon best sellers rank. https://www.amazon.com/gp/help/customer/display.html?nodeId=GGGMZK378RQPATDJ, 2022.

[9] Amazon. Understanding the amazon fling service. https://developer.amazon.com/docs/fling/understanding-the-amazon-fling-service.html, 2023.

[10] Syaiful Andy, Budi Rahardjo, and Bagus Hanindhito. Attack scenarios and security analysis of mqtt communication protocol in iot system. In *International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, 2017.

[11] Nordic Semiconductor ASA. nrf connect for mobile. https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp, 2023.

[12] Avast. Avast smart home security report 2019. https://cdn2.hubspot.net/hubfs/486579/avast_smart_home_report_feb_2019.pdf, 2019.

[13] Michael Backes, Sven Bugiel, and Erik Derr. Reliable third-party library detection in android and its security applications. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2016.

[14] Antonio Bianchi, Jacopo Corbetta, Luca Invernizzi, Yanick Fratantonio, Christopher Krügel, and Giovanni Vigna. What the app is that? deception and countermeasures in the android user interface. *In IEEE Symposium on Security and Privacy (S&P)*, 2015.

[15] Marco Casagrande, Eleonora Losiouk, Mauro Conti, Mathias Payer, and Daniele Antonioli. Breakmi: Reversing, exploiting and fixing xiaomi fitness tracking ecosystem. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022.

[16] Alex Chapman. Hacking into internet connected light bulbs. https://ajxchapman.github.io/security/2014/07/04/hacking-into-internet-connected-light-bulbs.html, 2014.

[17] Efstratios Chatzoglou, Georgios Kambourakis, and Christos Smiliotopoulos. Let the cat out of the bag: Popular android iot apps under security scrutiny. *Sensors*, 2022.

[18] Daming D. Chen, Maverick Woo, David Brumley, and Manuel Egele. Towards automated dynamic analysis for linux-based embedded firmware. In *Network and Distributed System Security Symposium (NDSS)*, 2016.

[19] Jiongyi Chen, Wenrui Diao, Qingchuan Zhao, Chaoshun Zuo, Zhiqiang Lin, Xiaofeng Wang, Wing Lau, Menghan Sun, Ronghai Yang, and Kehuan Zhang. Iotfuzzer: Discovering memory corruptions in iot through app-based fuzzing. In *Network and Distributed System Security Symposium (NDSS)*, 2018.

[20] Gareth Corfield. The regsiter. https://www.theregister.com/2021/10/21/iot_devices_corporate_networks_security_warning/, 2021.

[21] Bug Crowd. 1 crowdsourced cybersecurity platform bugcrowd. https://www.bugcrowd.com/, 2023.

[22] ct Open-Source. A collection of scripts to flash tuya iot devices to alternative firmwares. https://github.com/ct-Open-Source/tuya-convert, 2022.

[23] Ang Cui, Michael Costello, and Salvatore Stolfo. When firmware modifications attack: A case study of embedded exploitation. In *Network and Distributed System Security Symposium (NDSS)*, 2013.

[24] Erik Derr, Sven Bugiel, Sascha Fahl, Yasemin Acar, and Michael Backes. Keep me updated: An empirical study of third-party library updatability on android. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2017.

[25] dingjikerbo. Bluetoothkit—android bluetooth framework. https://github.com/dingjikerbo/Android-BluetoothKit, 2023.

[26] Shuaike Dong, Menghao Li, Wenrui Diao, Xiangyu Liu, Jian Liu, Zhou Li, Fenghao Xu, Kai Chen, XiaoFeng Wang, and Kehuan Zhang. Understanding android obfuscation techniques: A large-scale investigation in the wild. In *Security and Privacy in Communication Networks*. Springer International Publishing, 2018.

[27] Aneesh Dua, Vibhor Tyagi, ND Patel, and BM Mehtre. Iisr: A secure router for iot networks. In *International Conference on Information Systems and Computer Networks (ISCON)*, 2019.

[28] Zheng Fang, Hao Fu, Tianbo Gu, Pengfei Hu, Jinyue Song, Trent Jaeger, and Prasant Mohapatra. Iota: A framework for analyzing system-level security of iots, 2022.

[29] A Feal, Julien Gamba, Juan Tapiador, Primal Wijesekera, Joel Reardon, Serge Egelman, and Narseo Vallina-Rodriguez. Don't accept candy from strangers: An analysis of third-party mobile sdks. *Data Protection and Privacy, Volume 13: Data Protection and Artificial Intelligence*, 2021.

[30] Xiaotao Feng, Xiaogang Zhu, Qing-Long Han, Wei Zhou, Sheng Wen, and Yang Xiang. Detecting vulnerability on iot device firmware: A survey. *IEEE/CAA Journal of Automatica Sinica*, 2022.

[31] Organization for the Advancement of Structured Information Standards. Mqtt. https://mqtt.org/, 2022.

[32] Google. Malware categories. https://developers.google.com/android/play-protect/phacategories, 2022.

[33] Google. Shrink, obfuscate, and optimize your app. https://developer.android.com/studio/build/shrink-code, 2022.

[34] Grant Hernandez and Daniel Buentello. Smart nest thermostat a smart spy in your home. 2014.

[35] Muhammad Husnain, Khizar Hayat, Enrico Cambiaso, Ubaid U. Fayyaz, Maurizio Mongelli, Habiba Akram, Syed Ghazanfar Abbas, and Ghalib A. Shah. Preventing mqtt vulnerabilities using iot-enabled intrusion detection system. *Sensors*, 2022.

[36] Muhammad Ibrahim, Abdullah Imran, and Antonio Bianchi. Safetynot: On the usage of the safetynet attestation api in android. In *Proceedings of the Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2021.

[37] ilumi. New ilumi. https://play.google.com/store/apps/details?id=com.ilumibeta, 2023.

[38] Tuya Inc. Iot app sdk-tuya iot development platform. https://developer.tuya.com/en/docs/iot/app-sdk-instruction?id=K9kjstc7t376p, 2022.

[39] Belkin International. Looking for the wemo sdk? https://web.archive.org/web/20210224170725/http://developers.belkin.com/wemo/sdk, 2021.

[40] Yan Jia, Luyi Xing, Yuhang Mao, Dongfang Zhao, XiaoFeng Wang, Shangru Zhao, and Yuqing Zhang. Burglars' iot paradise: Understanding and mitigating security risks of general messaging protocols on iot clouds. In *IEEE Symposium on Security and Privacy (S&P)*, 2020.

[41] Yan Jia, Bin Yuan, Luyi Xing, Dongfang Zhao, Yifan Zhang, XiaoFeng Wang, Yijing Liu, Kaimin Zheng, Peyton Crnjak, Yuqing Zhang, Deqing Zou, and Hai Jin. Who's in control? on security risks of disjointed iot device management channels. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2021.

[42] Xin Jin, Sunil Manandhar, Kaushal Kafle, Zhiqiang Lin, and Adwait Nadkarni. Understanding iot security from a market-scale perspective. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2022.

[43] Arslan Khan, Hyungsub Kim, Byoungyoung Lee, Dongyan Xu, Antonio Bianchi, and Dave (Jing) Tian. M2MON: Building an MMIO-based security reference monitor for unmanned vehicles. In *USENIX Security Symposium*, 2021.

[44] Arslan Khan, Dongyan Xu, and Dave Jing Tian. Ec: Embedded systems compartmentalization via intra-kernel isolation. In *IEEE Symposium on Security and Privacy (S&P)*, 2023.

[45] Arslan Khan, Dongyan Xu, and Dave Jing Tian. Low-cost privilege separation with compile time compartmentalization for embedded systems. In *IEEE Symposium on Security and Privacy (S&P)*, 2023.

[46] Zhen Ling, Junzhou Luo, Yiling Xu, Chao Gao, Kui Wu, and Xinwen Fu. Security vulnerabilities of internet of things: A case study of the smart plug system. *IEEE Internet of Things Journal*, 2017.

[47] Pen Test Partners LLP. Steal your wi-fi key from your doorbell? iot wtf! https://www.pentestpartners.com/security-blog/steal-your-wi-fi-key-from-your-doorbell-iot-wtf/, 2022.

[48] Angela Lonzetta, Peter Cope, Joseph Campbell, Bassam Mohd, and Thaier Hayajneh. Security vulnerabilities in bluetooth technology as used in iot. *Journal of Sensor and Actuator Networks*, 2018.

[49] Shrirang Mare, Franziska Roesner, and Tadayoshi Kohno. Smart devices in airbnbs: Considering privacy and security for both guests and hosts. *Proceedings on Privacy Enhancing Technologies*, 2020.

[50] Yuhong Nan, Xueqiang Wang, Luyi Xing, Xiaojing Liao, Ruoyu Wu, Jianliang Wu, Yifan Zhang, and XiaoFeng Wang. Are you spying on me? large-scale analysis on iot data exposure through companion apps. 2023.

[51] Shradha Neupane, Faiza Tazi, Upakar Paudel, Freddy Veloz Baez, Merzia Adamjee, Lorenzo De Carli, Sanchari Das, and Indrakshi Ray. On the data privacy, security, and risk postures of iot mobile companion apps. In *Data and Applications Security and Privacy XXXVI*. Springer International Publishing, 2022.

[52] PhilipsHue. Philips hue sdk. https://github.com/PhilipsHue/PhilipsHueSDK-Java-MultiPlatform-Android, 2023.

[53] Marianna Rapoport, Philippe Suter, Erik Wittern, Ondrej Lhotak, and Julian Dolby. Who you gonna call? analyzing web requests in android applications. In *IEEE/ACM International Conference on Mining Software Repositories (MSR)*, 2017.

[54] Ole André V. Ravnås. Frida. https://frida.re/, 2022.

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

[56] Nilo Redini, Andrea Continella, Dipanjan Das, Giulio De Pasquale, Noah Spahn, Aravind Machiry, Antonio Bianchi, Christopher Kruegel, and Giovanni Vigna. Diane: Identifying fuzzing triggers in apps to generate under-constrained inputs for iot devices. In *IEEE Symposium on Security and Privacy (S&P)*, 2021.

[57] Chuangang Ren, Yulong Zhang, Hui Xue, Tao Wei, and Peng Liu. Towards discovering and understanding task hijacking in android. In *USENIX Security Symposium*, 2015.

[58] Majid Salehi, Danny Hughes, and Bruno Crispo. μSBS: Static binary sanitization of bare-metal embedded devices for fault observability. In *International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, 2020.

[59] Franziska Schwarz, Klaus Schwarz, and Reiner Creutzburg. Security and privacy investigation of wi-fi connected and app-controlled iot-based consumer market smart light bulbs. *Electronic Imaging*, 2021.

[60] Nordic Semiconductor. Nordic semiconductor. https://www.nordicsemi.com/, 2022.

[61] Nordic Semiconductor. nrf5 sdk. https://www.nordicsemi.com/Products/Development-software/nrf5-sdk/download, 2022.

[62] Nordic Semiconductor. Dual-bank and single-bank updates. https://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.sdk5.v12.0.0%2Flib_bootloader_dfu_banks.html, 2023.

[63] Nordic Semiconductor. nrf52840 dk. https://www.nordicsemi.com/Products/Development-hardware/nrf52840-dk, 2023.

[64] sensepost. objection - runtime mobile exploration. https://github.com/sensepost/objection, 2022.

[65] Lubomir Stroetmann and Tobias Esser. Reverse engineering the tp-link hs110. https://www.softscheck.com/en/reverse-engineering-tp-link-hs110/, 2018.

[66] SwitchBot. Switchbot smart switch button pusher. https://www.amazon.com/SwitchBot-switch-button-controlled-compatible/dp/B07B7NXV4R/, 2022.

[67] Microsoft Security Team. New security signals study shows firmware attacks on the rise; here's how microsoft is working to help eliminate this entire class of threats. https://www.microsoft.com/security/blog/2021/03/30/new-security-signals-study-shows-firmware-attacks-on-the-rise-heres-how-microsoft-is-working-to-help-eliminate-this-entire-class-of-threats/, 2021.

[68] The Wireshark team. Wireshark-go deep. https://www.wireshark.org, 2022.

[69] Positive Technologies. Positive technologies experts discover dangerous vulnerabilities in robotic vacuum cleaners. https://www.ptsecurity.com/ww-en/about/news/dangerous-vulnerabilities-in-robotic-vacuum-cleaners/, 2018.

[70] Telink. Telink — chips for a smarter iot. https://www.telink-semi.com/, 2022.

[71] Anurag Thantharate, Cory Beard, and Poonam Kankariya. Coap and mqtt based models to deliver software and security updates to iot devices over the air. In *International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2019.

[72] Daniel R. Thomas, Alastair R. Beresford, Thomas Coudray, Tom Sutcliffe, and Adrian Taylor. The lifetime of android api vulnerabilities: Case study on the javascript-to-java interface. In *Security Protocols XXIII*. Springer International Publishing, 2015.

[73] ThroughTek. Tutk sdk. https://github.com/cnping/TUTK, 2023.

[74] Ryan Tsang, Doreen Joseph, Asmita, Soheil Salehi, Nadir Carreon, Prasant Mohapatrak, and Houman Homayoun. Fandemic: Firmware attack construction and deployment on power management integrated circuit and impacts on iot applications. In *Network and Distributed Systems Security Symposium (NDSS)*, 2022.

[75] Tuya. Tuya smart - global iot development platform service provider. https://www.tuya.com/, 2022.

[76] Xueqiang Wang, Yuqiong Sun, Susanta Nanda, and XiaoFeng Wang. Looking from the mirror: Evaluating IoT device security through mobile companion apps. In *USENIX Security Symposium*, 2019.

[77] Belkin Wemo. Belkin wemo digital home automation. https://github.com/BelkinWemo/Wemo-Discovery-SDK, 2013.

[78] Haohuang Wen, Zhiqiang Lin, and Yinqian Zhang. Firmxray: Detecting bluetooth link layer vulnerabilities from bare-metal firmware. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2020.

[79] Inc Wonderlabs. Switchbot. https://play.google.com/store/apps/details?id=com.theswitchbot.switchbot, 2022.

[80] Jianliang Wu, Ruoyu Wu, Daniele Antonioli, Mathias Payer, Nils Ole Tippenhauer, Dongyan Xu, Dave (Jing) Tian, and Antonio Bianchi. LIGHTBLUE: Automatic Profile-Aware debloating of bluetooth stacks. In *USENIX Security Symposium*. USENIX Association, 2021.

[81] John Wu. Magisk. https://github.com/topjohnwu/Magisk, 2022.

[82] Wei Xie, Yikun Jiang, Yong Tang, Ning Ding, and Yuanming Gao. Vulnerability detection in iot firmware: A survey. In *IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, 2017.

[83] Huikai Xu, Miao Yu, Yanhao Wang, Yue Liu, Qinsheng Hou, Zhenbang Ma, Haixin Duan, Jianwei Zhuge, and Baojun Liu. Trampoline over the air: Breaking in iot devices through mqtt brokers. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2022.

[84] Muneer Bani Yassein, Mohammed Q. Shatnawi, Shadi Aljwarneh, and Razan Al-Hatmi. Internet of things: Survey and open issues of mqtt protocol. In *International Conference on Engineering & MIS (ICEMIS)*, 2017.

[85] Koen Zandberg, Kaspar Schleiser, Francisco Acosta, Hannes Tschofenig, and Emmanuel Baccelli. Secure firmware updates for constrained iot devices using open standards: A reality check. *IEEE Access*, 7, 2019.

[86] Zensys. Z-wave. https://www.z-wave.com, 2020.

[87] Yiwei Zhang, Siqi Ma, Juanru Li, Dawu Gu, and Elisa Bertino. Kingfisher: Unveiling insecurely used credentials in iot-to-mobile communications. In *Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2022.

[88] Zhi-Kai Zhang, Michael Cheng Yi Cho, Chia-Wei Wang, Chia-Wei Hsu, Chong-Kuan Chen, and Shiuhpyng Shieh. Iot security: ongoing challenges and research opportunities. In *IEEE international conference on service-oriented computing and applications*, 2014.

A. Additional Case Study: JcPrinter

We identified *JcPrinter-SDK* in the NIIMBOT companion app from the *DeviceDataset*. We analyzed the NIIMBOT D11 Label Printer that is controlled by the NIIMBOT app. After analyzing the NIIMBOT app using our *AoT-Scout* analysis, we discovered that *JcPrinter-SDK* is handling Bluetooth communication with the Label Printer. After connecting to the companion app, the Label Printer sends the current firmware version running on the Label Printer along with other information about the device to the NIIMBOT app. The NIIMBOT app forwards this information to the app's backend server. The backend server responds with information in accordance with the information sent by the NIIMBOT app. If the device is on the latest version, the backend server provides no information about the updated firmware binary. The Label Printer we analyzed is on the latest version, so we got no information about the firmware binary from the backend server. To trigger the DFU process, we spoofed the firmware version sent by the Label Printer to the NIIMBOT app. We spoofed the firmware version number by dynamically injecting code in the companion app using Frida and advertising a lower firmware version number lower than the actual firmware version number. We were able to trigger the DFU mechanism after firmware version spoofing. After triggering the DFU mechanism, we retrieved the URL of the updated firmware binary from the backend server. We downloaded the firmware binary using the URL provided by the backend server.

After acquiring the firmware binary, our goal was to test the firmware modification attack by sending a modified firmware binary to the Label Printer. To this aim, we uploaded the modified firmware binary to our server. Then we triggered the DFU process again by spoofing the firmware version number. When the NIIMBOT app's backend server responded with the updated binaries URL, we intercepted the backend server's response and replaced the URL of the updated firmware binary with the URL of our server that has the modified firmware binary. Now the NIIMBOT app downloaded the modified firmware from our server.

After the NIIMBOT app downloaded our modified binary, *JcPrinter-SDK* computed the CRC of the binary and verified the integrity of the binary. The NIIMBOT app performed CRC verification on the binary and stopped the DFU process after it detected the modification. We bypassed this CRC verification by dynamically injecting code into the NIIMBOT app using Frida. After bypassing the CRC verification, we were able to complete the DFU process and successfully transferred the modified firmware to the Label Printer. We confirmed that the modified firmware was sent to the Label Printer by sniffing the Bluetooth traffic and verifying that the bytes sent to the Label Printer matched the bytes of our modified firmware. Finally, to confirm the success of the firmware modification attack, we ensured that the Label Printer ran the modified firmware. To this aim, we verified that the firmware version of the modified binary was equal to the firmware version sent by the Label Printer. Since the *JcPrinter-SDK* is a proprietary SDK and is only used by one companion app, we did not verify the firmware modification attack on an additional device.

B. Additional Tables

Table 2 lists the Amazon Bestseller Categories we used to obtain our initial list of IoT devices (as explained in Section 5.1). Table 3 provides details of all the devices analyzed in this paper.

TABLE 2: Amazon Bestseller Categories.

Home Automation Hubs and Controllers	
GPS, Finders and Accessories	
Home Automation	Smart Baby Monitors
Electrical Outlet Switches	Electronics
Indoor Thermometers	Devices and Accessories
Baby Monitors	Baby Sleep Soothers
Camera and Photo Products	LED Bulbs
Nursery Night Lights	Sleep Sound Machines
LED Strip Lights	Computer Printers

Device Name (Device Type)	App Package Name (Version Name), (Version Code)	App Name	SDK	Dataset
Aoycocr Bluetooth WiFi Smart Plug (Plug)	com.xiaozhanlianbing (1.0.2), (4)	Popotan	Tuya [38]	Amazon
ESICOO Smart Plug (Plug)	com.aliyun.iot.living (3.7.0), (370)	Cloud Intelligence	Aliyun [4]	Amazon
Wemo Smart Plug WSP080 (Plug)	com.belkin.wemoandroid (1.2), (160)	Wemo	Wemo [77]	Amazon
Kasa Smart Plug Mini KP115 (Plug)	com.tplink.kasa_android (2.33.1.998), (998)	Kasa Smart	Tplinkra (P)	Amazon
Amazon Smart Plug (Plug)	com.amazon.dee.app (2.2.416420.0), (892952711)	Amazon Alexa	Alexa (P)	Amazon
Tracki 2022 4G LTE Mini GPS (Tracker)	com.trackimo.android.tracki (1.0.199), (202108021)	Tracki GPS – Track Cars, Kids,	AltBeacon [7]	Amazon
Tile Mate (2022) Bluetooth (Tracker)	com.thetileapp.tile (2.82.0), (3522)	Tile: Making Things Findable	Tile (P)	Amazon
Apple AirTag (Tracker)	com.apple.trackerdetect (1.1), (8)	Tracker Detect	*OBF* (P)	Amazon
LandAirSea 54 GPS (Tracker)	com.landairsea.silvercloud (4.29), (129)	SilverCloud	LandAirSea (P)	Amazon
Tenmiro LED Strip (Light)	com.zjf.kslight (1.2.6), (126)	KeepSmile	Inuker [25]	Amazon
Daybetter LED Strip (Light)	com.qh.Apollo (1.004), (4)	Apollo Lightning	Consmart (P)	Amazon
Wyze Cam v2 1080p Indoor (Camera)	com.hualai (2.20.21), (61532)	Wyze	Tutk [73]	Amazon
Blurams Camera Dome Lite 2 (Camera)	com.blurams.ipc (5.1049.0.283), (1283)	blurams	*OBF* (P)	Amazon
Govee Hygrometer Thermometer H5075 (Environment Sensor)	com.govee.home (4.4.3), (214)	Govee Home	Govee (P)	Amazon
SwitchBot Smart Switch (Switch)	com.theswitchbot.switchbot (5.2.6.6), (211)	SwitchBot	NordicSecure [61]	Amazon
NIIMBOT D11 (Printer)	com.gengcon.android.jccloudprinter (4.5.1), (376)	NIIMBOT	JcPrinter (P)	Amazon
Hatch Rest Mini Sound Machine (Speaker)	com.hatchbaby.rest (3.0.3), (699)	Hatch Sleep	HatchBaby (P)	Amazon
Amazon Fire TV Stick (Media Player)	com.amazon.storm.lightning.client.aosp (1.0.18.00), (1000180000)	Amazon Fire TV	Fling [9]	Amazon
Philips Hue A19 LED (Light)	com.signify.hue.blue (1.31.0), (3852)	Philips Hue Bluetooth	Hue/Signify [52]/(P)	Amazon
SYLVANIA Bluetooth Mesh LED 75763 (Light)	com.ledvance.smartplus (2.2.29), (2020029)	SYLVANIA Smart Home	Telink [70]	IoTSpotter
ilumi Bluetooth Smart LED A19 (Light)	com.ilumibeta (4.3.30), (100)	New ilumi	Nordic [61]	Verification
NUT Key Finder (Tracker)	com.nut.blehunter.findthing (3.11.36), (20210522)	Findthing - Smart Finder	NordicSecure [61]	Verification
GoSund Power Strip (Switch)	com.gosund.smart (3.22.5), (15)	Gosund	Tuya [38]	Verification

TABLE 3: List of all devices analyzed in this paper. (P) = Proprietary SDKs. *OBF* = Obfuscated.