Living in the Past: Analyzing BLE IoT Devices Based on Mobile Companion Apps in Old Versions

Jianqi Du*†, Zidong Zhang*†, Fenghao Xu‡§(✉), and Wenrui Diao*†(✉)
*School of Cyber Science and Technology, Shandong University
{dujianqi, keelongz}@mail.sdu.edu.cn, diaowenrui@link.cuhk.edu.hk
†Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University
‡Southeast University, xf016@link.cuhk.edu.hk §The Chinese University of Hong Kong

***Abstract*—Bluetooth Low Energy has been a widely adopted communication technique in the consumer IoT market. Meanwhile, the security concerns of these BLE-enabled IoT devices have garnered considerable attention. Instead of investigating the device firmware directly, analyzing its companion mobile app has been proven to be an effective approach for vulnerability discovery. However, developers regularly release new versions of these apps, making it more challenging to analyze and identify vulnerabilities. As a result, this action raises the bar on launching attacks on IoT devices. In our study, we found that the earlier versions of the companion apps can still be exploited to attack IoT devices. The key insight is that these devices usually lack firmware update capabilities.**

In our work, we performed attacks on three BLE-enabled IoT devices by investigating the early versions of their companion apps. We observed that manufacturers merely updated the companion apps to increase the difficulty of reverse engineering through code protection techniques without addressing the vulnerabilities presented in the device firmware. We then conducted a large-scale measurement and confirmed that most BLE devices can be analyzed from their old app versions. Furthermore, we design an automated tool to help developers identify the risks and improve the security of their apps. In our study, we also discuss some mitigation solutions.

***Index Terms*—IoT security, firmware update, Bluetooth low energy, Android app analysis**

I. Introduction

Bluetooth Low Energy (BLE) was introduced by the Bluetooth Special Interest Group (SIG) as a low-energy alternative to the classical Bluetooth technology in Bluetooth 4.0 [2]. Currently, it has become an important technology in the development of IoT devices. These devices typically communicate with a central control unit or a user's smartphone through a companion app. The companion app acts as an interface for users to control, monitor, and configure the IoT device, making it a critical component of the IoT system.

Although companion apps provide convenience, they also come with a price. The security of these apps is often overlooked, creating an exploitable interface for attackers to compromise IoT devices [28]. For example, many companion apps communicate with IoT devices over BLE, some of which may lack proper encryption and authentication mechanisms. Attackers can intercept and manipulate data exchanged between the app and the device through BLE, potentially leading to unauthorized control or theft of sensitive information. Therefore, companion apps must stay up to date with the latest security updates and improvements. When vulnerabilities are discovered in the companion app, developers can release new versions of the app on the app market. Upgraded versions may utilize code protection techniques to protect their source code and prevent attackers from reverse engineering. Unfortunately, manufacturers and developers sometimes neglect older versions of those apps [36], leaving them vulnerable to malicious exploitation. Hackers capitalize on this weakness because outdated apps may lack essential security protection, making it easier for attacks such as unauthorized access.

The most effective way to mitigate the threat is to update the IoT firmware with security patches via Over-The-Air (OTA). However, we found that most IoT devices do not have the capability to update their firmware. There are many reasons for the absence of OTA capability. It could be that the manufacturer chose a BLE chip without OTA functionality or did not develop firmware upgrade functionality in their companion app.

When firmware updates are impossible, developers and vendors resort to using code protection techniques to prevent reverse engineering of new app versions. Although these countermeasures make apps difficult to analyze, vulnerabilities still exist in the firmware of IoT devices. Based on this observation, we found that old versions of companion apps are more easily exploited to attack IoT devices, especially those IoT devices that lack firmware update capability.

In this paper, our findings can help to efficiently identify vulnerabilities in IoT devices. To demonstrate the effectiveness of our findings, we provide three motivation examples. They primarily show the successful exploitation of real-world IoT devices using the old version. Motivated by the above cases, we conducted a large-scale experiment on a dataset of 37,778 IoT apps and filtered out 11,045 BLE apps for further analysis. It is astonishing that over 81% of BLE-enabled IoT devices could potentially be influenced by this. The measurement results show that our method has a wide range of applications. We then developed a tool named BLESecurAssist to help developers or users identify potential risks to IoT devices. Our tool takes an app as input and automatically downloads its older versions. In addition, it performs an automated analysis of these outdated apps and outputs potential vulnerabilities in the associated IoT devices.

Contributions. In summary, our work makes the following contributions.

- *New Findings.* We found that analyzing the early versions of IoT companion apps is easier and more efficient, which can assist in the discovery of vulnerabilities on IoT devices. The key insight is that BLE devices usually lack firmware update capabilities.
- *Measurement and Case Study.* To demonstrate the impact of our findings, we performed a large-scale measurement on an IoT app dataset (37,778). The result shows that we can trace most BLE apps' older versions, facilitating a more practical security analysis.
- *Tool Design and Discussion.* We further integrate our measurement technique into an analysis tool that could help developers or users identify the potential risks of IoT devices.

Roadmap. The rest of this paper is organized as follows. Section II provides the necessary background. In Section III, based on the threat model and identified security risks, we present three motivating examples. The results are summarized in Section IV. Section V gives the detailed design of BLESecurAssist. Mitigation suggestions are discussed in Section VI. Section VII reviews the related works, and Section VIII concludes this paper.

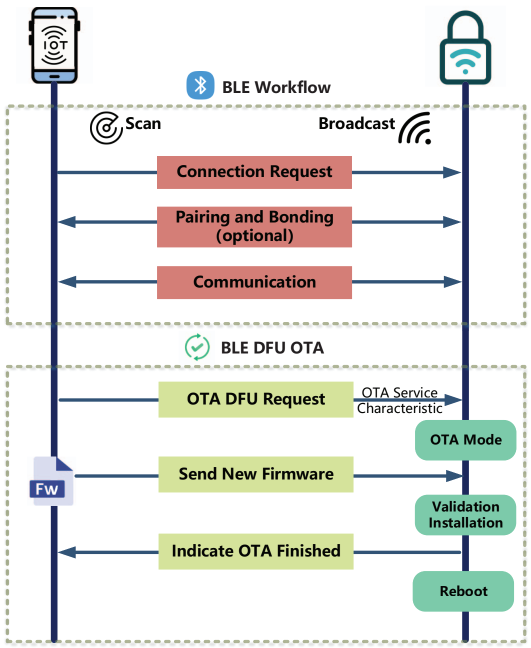

Fig. 1: BLE Workflow and OTA Process

II. Background

Bluetooth Low Energy (BLE) is a wireless communication technology designed for low-power data transmission over short distances. This section provides an overview of the working principles of BLE, the concept of UUID (Universally Unique Identifier) in BLE, and the OTA firmware update process in BLE. Figure 1 illustrates the BLE workflow, including the OTA firmware update process.

BLE Workflow. In the design of BLE, the typical communication between two devices involves three stages: (1) Advertising, (2) Scanning, and (3) Communication. The BLE workflow can be summarized as follows:

1) Advertising: BLE devices use advertising packets to broadcast their presence and capabilities to potential neighboring devices. The advertising packets contain essential information for device discovery, such as device name, service UUIDs, manufacturer-specific data, service data, and signal strength indicator (RSSI). The device name allows for human-readable identification, while the UUID aids in distinguishing the device and its supported services. Manufacturer-specific data and service data provide opportunities for customizing advertising content that satisfies specific application requirements.

2) Scanning: The scanning process is a crucial operation in the BLE protocol, enabling devices to discover and identify nearby devices through the reception and interpretation of advertising packets. Upon receiving an advertising packet, the scanning device can extract and process the information it contains. This information includes the device's identification details, signal strength, and other relevant data. The scanning process facilitates the creation of a local database of nearby BLE devices, allowing the scanning device to determine potential devices with which it may establish a connection.

3) Communication: After the scanning device finds a BLE device of interest, it can initiate a connection with it. BLE supports two types of connections: "Central" and "Peripheral". A Central device, like a smartphone or a laptop, initiates the connection to a Peripheral device, such as a sensor or a wearable device. Once the connection is established, these devices can exchange data bi-directionally. BLE communication follows a client-server architecture, which means that the Central device acts as the client, and the Peripheral device acts as the server. Data exchange typically occurs through "characteristics" and "services", which are defined in the Generic Attribute Profile (GATT). GATT specifies how data can be structured and accessed between connected BLE devices, allowing custom data transfer and device control.

BLE Security Measures. As BLE technology continues to gain popularity, ensuring security is paramount. BLE incorporates various security features to protect sensitive data and prevent unauthorized access. Some key security measures include:

1) Encryption: BLE devices can encrypt data during communication to ensure confidentiality. The encryption process uses a shared key established during the pairing phase.

2) Pairing: Before establishing a connection, BLE devices could undergo a pairing process. During pairing, devices exchange security information and establish a trusted relationship.

3) Privacy: BLE devices utilize a privacy feature that can periodically change the device's Bluetooth MAC address, making it harder for unauthorized parties to identify and track the device over time.

BLE OTA Capability. BLE OTA Firmware Update is a mechanism that enables wireless firmware update on BLE-enabled devices. It allows device manufacturers to remotely and securely push new firmware versions to devices, ensuring they stay up-to-date with the latest features, bug fixes, and improvements without needing physical connections. Some key steps of the OTA firmware update are as follows:

1) OTA Mode Activation: The device must have a mechanism to enter the OTA mode. This can be triggered through a specific command, a button combination, or a remote BLE connection. In OTA mode, the device is ready to receive the new firmware.

2) Firmware Validation: During the download process, the target device may perform integrity checks and verify the authenticity of the incoming firmware to ensure it hasn't been tampered with during transmission. This step is crucial for ensuring security and preventing unauthorized firmware updates.

3) Firmware Installation: Once the complete firmware is downloaded and verified, the target device proceeds with installation. After the new firmware is successfully installed, the target device typically performs a system reset or reboot to activate the updated firmware.

III. Threat Model and Motivating Examples

In this section, we first outline the threat model that our work is based on. We then introduce the motivation of our research by presenting three real-world cases based on the threat model. These cases aim to highlight the key insights of our approach.

A. Threat Model

In this paper, we assume that the attacker has the following practical capabilities. The nearby attacker can sniff all BLE packets transmitted over the air and have physical access to the BLE device. An attacker can obtain the IoT device's early version of the app. The attacker can extract control commands or other sensitive information from the poorly protected early versions of apps. In the threat model, the attacker mainly uses eavesdropping and replay attacks to gain unauthorized access to IoT devices. These attacks can be launched using a Bluetooth dongle, such as the CC2540 sniffer from Texas Instrument [4].

B. Motivating Examples

In this section, we discuss attack instances targeting three BLE-enabled IoT devices. These devices are vulnerable to exploitation through old versions of their companion apps. To help better understand, Figure 2 shows the attack flow in the first case.

1) Smart Lock. OKLOK is a smart lock vendor with thousands of users of its products. Their BLE smart lock

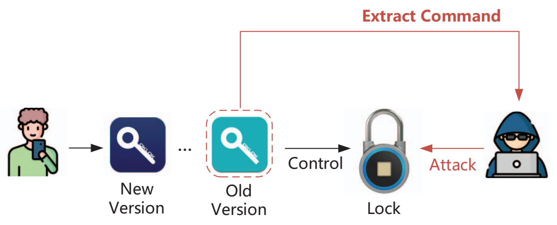

Fig. 2: Attack Flow.

enables users to unlock and manage permissions through a companion app on their mobile devices.

To investigate the authorization between OKLOK's lock and its app, we first downloaded the latest companion app from the Android app market. After we decompiled the app with the state-of-the-art reverse tool JADX [9], we found that the app was developed with the uni-app, a framework that uses WebView to create mobile apps. Through further analysis, we found that the manufacturer's cloud servers host the core code in the app, and the interaction is implemented in the native code of the app. In other words, we cannot restore the authorization process by reversing the app since we cannot access the source code.

However, we discovered that an older version of the companion app could also control the same lock, so we turned our attention to the app in the early version for this lock. After obtaining the earlier version from the Wandoujia [21] App market, we performed a manual analysis. We found that the unlock function was developed in Java without any code protection. As a result, it can be easily reverse-engineered and analyzed. Figure 3 illustrates the unlocking mechanism. When the user presses the unlock button in the app, the mobile client will send an open request to the smart lock via BLE. After the lock receives the request, it sends a nonce as a token to the mobile. Then, the app will utilize the token and other information to construct a valid open command, which will be sent as a response to the lock. If they are equal, the lock will open. In summary, this process is typically a Challenge-Response authentication.

To reproduce the above authorization process, we used a BLE dongle to connect the locks and inspect packet communication between the communication. After obtaining the random token from the lock, we generated a valid command by following the process we reversed from the app. After we sent the generated command to the lock, we received a success message from the lock and found that it was unlocked. Thus, we succeeded in gaining unauthorized access to the lock by analyzing the old version of the companion app.

2) Smart Bulb. LifeSmart provides a wide range of IoT smart devices and allows users to control and manage them with a companion app. Users can use the accompanying app to switch it on and off, change colors, and other functions.

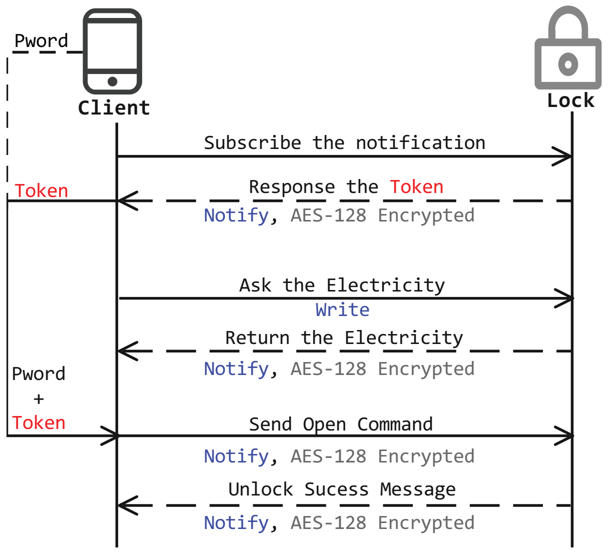

Fig. 3: The Unlock Process of a Smart Lock.

Like OKLOK, we also want to analyze the communication between the smart bulb and the client through the accompanying app. When we got the latest version of the companion app from the official Google Marketplace, we found that the app kept all its core code in `.so` library files in the form of native code. In other words, it is not easy to analyze these files to obtain a trace of the BLE communication process to control the bulb.

Inspired by OKLOK's case, we downloaded the old version of LifeSmart's companion app from Wandoujia. By manually analyzing, we found that it can easily be reversed. After restoring their BLE communication flow, we could utilize a BLE dongle to connect with the bulb and construct the corresponding commands to achieve unauthorized control over it.

3) Smart WristBand. Our third attack target was a smart band called Lefun, which is a wearable device that allows users to monitor their daily physical activities and fitness levels. The band uses BLE communication to transfer data and sync with its companion app, including mobile message synchronization. We found that when the phone synced a long message, it would be truncated before displaying on the band. This inspired us to see if we could bypass this mechanism and crash the bracelet by sending a malformation message.

After analyzing the latest app, we found that it utilized code obfuscation techniques, making it challenging to locate the target code. Fortunately, we found that it did not use any code protection in the early version of the app, which allowed us to reverse the target code and forge a valid message transmitted to the band easily.

Through further analysis, we found that excessively long messages would complete truncation in the app before being sent to the band. However, the band crashed after we used the BLE dongle to send a valid over-length message to it, which proves that the band's firmware does not check the message length, leading to the overflow.

IV. Problem and Analysis

A. Research Questions

By observing the aforementioned attack examples, it inspires us to figure out why this approach is effective. Thus, we propose the following research questions and provide our insights on each.

Q1) Why are the vulnerabilities discovered in the early versions still exploitable? The vulnerabilities discovered in early versions remain effective for IoT devices mainly because manufacturers have not fixed the vulnerabilities in the device firmware side. However, most IoT device firmware lacks the ability to upgrade, so these vulnerabilities will hardly be fixed.

Q2) Why are the vulnerabilities more easily discovered in early versions than in upgraded versions? The early app versions will use more code protection techniques to prevent reverse engineering of their source code. However, earlier versions usually did not pay much attention to the security aspect during development, making it easier for attackers to analyze the code.

Based on the two research questions, we conduct further in-depth analysis. Regarding Q1, it is crucial to ascertain the extent to which the security of IoT devices is affected by the early app versions. Therefore, we conduct a large-scale measurement on the firmware updating capabilities for BLE-enabled IoT devices. As for Q2, we seek to understand what kind of code protection techniques are employed in the upgraded app versions. Hence, we need to acquire a collection of companion apps for IoT devices, including both their latest and older versions. Subsequently, we will conduct a comparative analysis to determine the code protection techniques employed by manufacturers. This will enable us to gain a comprehensive understanding of the progress and improvements made by different manufacturers in terms of app security.

B. Firmware Update Analysis

In this section, we conduct a large-scale measurement on the firmware updating capabilities for BLE-enabled IoT devices. Our experiment consists of the following steps:

1) Obtain a dataset of BLE devices' companion APKs.
2) Collect OTA update strategies for different BLE SoCs.
3) Perform automatic analysis and output results.

BLE-Enabled IoT App Dataset. Since no large-scale dataset is available for BLE-enabled IoT Android apps, we focus on downloading these apps from the AndroZoo [22]. However, Androzoo does not provide any information indicating an app's classification as a BLE-enabled IoT application. The first challenge is to determine how to identify these apps as IoT apps. Jin et al. [27] proposed a framework named IoTSpotter, which automatically constructs a market-scale snapshot of mobile-IoT apps, and also provided an IoT app dataset filtered from the AndroZoo. We decided to utilize the IoT app dataset and filter out the IoT apps related to BLE. Firstly, we begin by filtering out apps that declare Bluetooth permissions (i.e., BLUETOOTH and BLUETOOTH_ADMIN). As apps filtered based on Bluetooth permissions may contain Bluetooth Classic and BLE, we subsequently utilize BLE-related classes (e.g., android.bluetooth.BluetoothGatt) as an additional criterion to further refine the filtering process for identifying BLE apps. It should be noted that first filtering by Bluetooth permissions and then filtering by the BLE libraries can effectively improve the overall process speed. The advantage lies in the fact that permission checking relies on the app Manifest files without the need to disassemble code.

BLE SoCs OTA. Bluetooth Low Energy System-on-Chips (BLE SoCs) are small, low-power, and highly integrated solutions that combine the functionalities of a microcontroller, a Bluetooth radio, and other peripherals into a single chip. BLE SoCs are available in numerous models and brands, with distinct features and application range. Selecting the appropriate BLE SoC depends on the project's specific requirements, such as power consumption, processor performance, OTA update, and other functionalities. OTA updates allow devices to receive firmware or software updates wirelessly without the need for physical connections. There are several manufacturers that produce SoCs that support OTA functionality [19], such as Nordic Semiconductor, Texas Instruments, and so on. Through the manufacturers' provided instructions and guidelines, we find that these SoCs support OTA updates using a specific UUID. The UUID serves as a unique identifier for the OTA service, which facilitates firmware or software updates wirelessly over the BLE. Therefore, if an IoT vendor wants to update their devices' firmware through the companion app, it needs to hardcode the specific UUIDs in their app.

We have collected firmware upgrade strategies from ten leading Bluetooth chip suppliers with the highest market share in the Bluetooth industry. They offer the most comprehensive SDKs and documentation. Table I lists the OTA update service UUIDs by SoCs vendors.

To extract all static UUIDs hardcoded within the APK file from the dataset, we employ Androguard [1] for analyzing apps. Androguard is a widely used open-source tool specifically designed for static analysis of Android apps. After completing the extraction of UUIDs for each app, we further examine these UUIDs to determine if there are any matches with the OTA-related UUIDs we previously collected. If we discover such OTA-related UUIDs within the App, it confirms that the corresponding IoT device for this App is capable of firmware upgrades.

Environment setup. Our evaluation was performed on a Linux server running Ubuntu 20.04, powered by an Intel Xeon 6226R @ 2.90GHz processor and with 256GB of memory.

Measurement Result. We downloaded 37,778 IoT apps from Androzoo using the SHA256 indexes provided by Jin et al. From the dataset, we filtered out 11,045 BLE-enabled IoT apps. Among these, 2,039 apps contained at least one UUID related to OTA. Therefore, more than 81% of IoTs cannot support OTA upgrades. Figure 4 shows the average ratio of SoCs usage.

TABLE I: Firmware Update UUIDs

Manufacturer	OTA Service UUID(s)
Nordic[10]	0x0000FE59-0000-1000-8000-00805F9B34FB 0x8E400001-F315-4F60-9FB8-838830DAEA50 0x00001530-1212-EFDE-1523-785FEABCD123
Texas[20]	0xF000FFC0-0451-4000-B000-000000000000
SiliconLabs[15]	0x1D14D6EE-FD63-4FA1-BFA4-8F47B42119F0
Renesas[13], [14]	0x9D5998F8-105B-4691-92BE-4B1B4D3EE8BB 0x00010203-0405-0607-0809-0A0B0C0D1912
STM[16], [17]	0x0000FE20-CC7A-482A-984A-7F2ED5B3E58F 0x8A97F7C0-8506-11E3-BAA7-0800200C9A66
NXP[11]	0x0000FEE8-0000-1000-8000-00805F9B34FB
Cypress[7]	0x00060000-F8CE-11E4-ABF4-0002A5D5C51B
Espressif[8]	0x00008018-0000-1000-8000-00805F9B34FB 0x1775244D-6B43-439B-877C-060F2D9BED07
Qualcomm[5], [6]	0x00001100-D102-11E1-9B23-00025B00A5A5 0x00001016-D102-11E1-9B23-00025B00A5A5
Telink[18]	0x00010203-0405-0607-0809-0A0B0C0D1912
Realtex[12]	0x00006287-3C17-D293-8E48-14FE2E4DA212 0x0000D0FF-3C17-D293-8E48-14FE2E4DA212

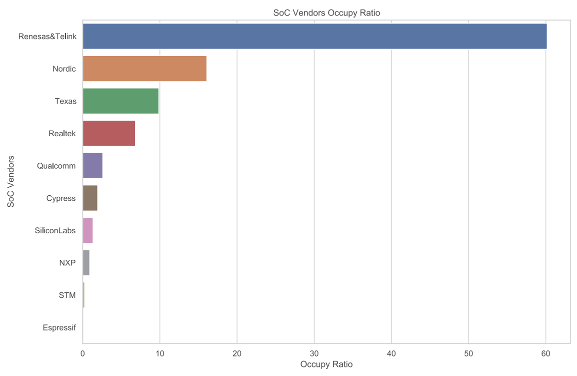

Fig. 4: BLE SoC Vendors Occupy Ratio

C. Old Version and Code Protection

To provide a more comprehensive response to Q2, we conduct a comparative analysis between the older and newer versions of IoT companion apps. This comparative study aims to figure out the techniques implemented by manufacturers in the new app version to make it more resistant to vulnerability detection compared to the older version. To analyse the old versions of companion apps, the first challenge is how to obtain them. One legitimate method is to download from third-party app stores that host older versions of apps. Wandoujia was a popular third-party Android app market in China. The most unique feature of Wandoujia compared with other third-party app marketplaces is that it provides the download of historical versions of an app.

Once the relevant apps were downloaded from Wandoujia, our next step was to select a few of them for manual analysis. To better understand distinctions among these applications, we used the JADX tool, specially designed for reverse engineering. By decompiling the APKs, we gained access to the Java code of both the old and new versions of the apps. The subsequent stage involved conducting a comparative analysis of the Java code between the older and recent iterations. We have outlined the evolution of these companion apps in the following.

- **Code Obfuscation.** Code obfuscation [24] is a process of making code harder to read or understand without changing its functionality. It is often used in Android app development to protect the code from reverse engineering and unauthorized access. We found through comparing old and new versions that this is the most widely adopted code protection technology.
- **App Packing.** In software development and security, App Packing (Encryption) is a common security measure. During our analysis, We found that this technology was generally not adopted in the old version. We consider that this is due to the higher performance requirements of the phone in older versions.
- **Java Native Interface.** Java Native Interface (JNI) is a framework that enables Java code to call and be invoked by native apps and libraries written in other languages, such as C, C++, and assembly. We found that in the new version, programmers use JNI technology to protect the code, while in the old version, it was generally written in Java code.
- **Webview.** WebView is a core component of the Android operating system that allows developers to embed web content (such as HTML, CSS, and JavaScript) into their Android apps. We found that in the old version, a lot of data was statically encoded within the APK, while in the new version, data is loaded from the manufacturer's server. Undoubtedly, this has increased the difficulty of static analysis of the App.

In Android app development, code protection is critical to prevent unauthorized access and reverse engineering of an app's source code. That's why finding vulnerabilities in older versions is easier than newer ones.

V. Design of BLESecurAssist

Based on the previous analysis, we have implemented a framework – BLESecurAssist that integrates our measurement technique. The purpose of developing this framework is to assist developers or users in identifying potential risks of their IoT devices. Moreover, the framework can identify BLE misconfigurations in IoT devices.

The framework comprises three major components: old versions downloader, OTA analysis, and misconfiguration detection. The framework accepts the latest version of the companion app and returns the possibility of attacking this IoT device. Figure 5 provides a comprehensive overview of the framework's architecture.

Old Versions Downloader. In our earlier analysis, we downloaded older versions of apps from the Wandoujia app market. However, we discovered that Wandoujia discontinued the service for accessing historical versions on June 15, 2023. To overcome this challenge, we have proposed a solution.

We developed a fundamental information extractor. It can extract essential information about the APK, such as package name and version code. First, we need to decompile the APK using Androguard's built-in capabilities. Once decompiled, locate the AndroidManifest.xml file within the extracted files. Within this XML file, it can easily identify the package name and version code attribute. After extracting the attribute, we turn to AndroZoo, which offers a substantial CSV file (over 2.7GB compressed) containing more than 23 million records, with fields like "pkg_name" and "vercode". These fields represent the name of the Android Package and the version code, respectively.

To proceed with our analysis, we filter out entries in the CSV file that share the same package name as the new version of the app but have different version codes. These entries correspond to the older versions we are interested in, and we can download the old versions through the API provided by Androzoo. By employing this approach, we can continue our research on the historical versions of the apps, even after the discontinuation of the service by Wandoujia. This method allows us to gather valuable data and insights and explore the evolution and changes in these companion apps over time.

OTA Analysis. After resolving the issue of downloading old versions of apps, we are confronted with another critical question: to determine whether these older versions of apps can still effectively control IoT devices. This matter is particularly significant because if certain IoT devices lack OTA update capabilities, it implies that older versions of apps can still be used to control these devices. Consequently, any vulnerabilities discovered within these apps can potentially be exploited to launch attacks on IoT devices.

To verify the OTA capability of these apps, we gathered OTA update solutions from major chip manufacturers and utilized the Androguard tool to extract and analyze UUIDs and DFU libraries within the apps. If we fail to identify any OTA-related data in these different versions of the apps, we can deduce that the corresponding IoT devices do not possess OTA capabilities.

The difficulty in obtaining IoT firmware due to the reluctance of major manufacturers to provide firmware downloads poses a challenge in analyzing IoT device security. To aid security developers in conducting a thorough analysis of IoT device security from the firmware, our tool can automatically extract the firmware of IoT devices (if available) from APKs, provided we find that the corresponding IoT devices have OTA capabilities in the app. This is because some manufacturers package their firmware within APKs for device upgrades.

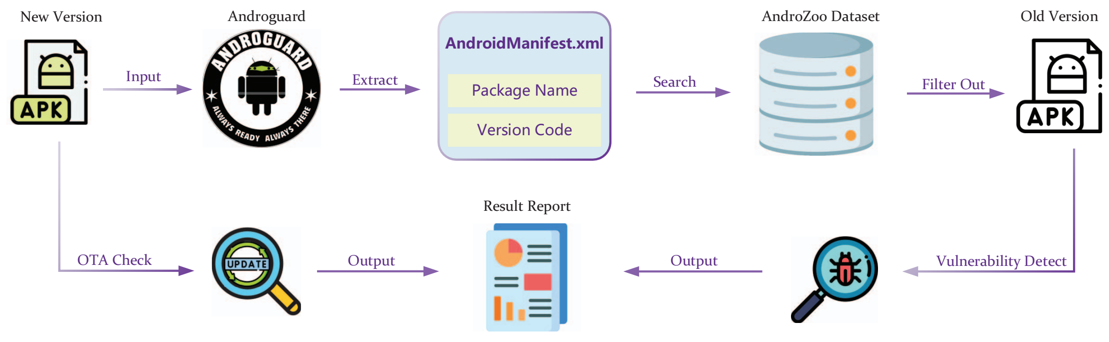

Fig. 5: Analysis Flow

Using this feature, we successfully extracted 966 firmware images from 11,045 BLE-enabled IoT apps. Through this research and tool development, we aim to provide convenience in analyzing IoT device security and enable a more comprehensive assessment of their safety.

Mis-configurations Detection. To ensure the security of these devices is crucial to prevent potential attacks and protect user data, BLE specifications [3] provide security guidelines for data encryption, authentication, and authorization during communication. Unfortunately, if developers fail to adhere to these specifications, it can introduce security vulnerabilities in IoT devices. In our framework, we incorporate the detection process for misconfigurations in BLE apps. We could detect static Bluetooth MAC address or UUIDs (can be abused for device tracking), insecure pairing method, plaintext data transmission, and so on. For example, a previous work BSC-Checker [25] can be applied in our framework to identify the above issues.

Using such tool, IoT developers can detect and fix security weaknesses in their BLE apps, reducing attack surfaces and protecting sensitive data transmitted over BLE connections.

VI. Discussions

Lessons Learned and Suggestions. If the manufacturer's Bluetooth chip supports OTA functionality, we encourage them to promptly develop firmware upgrade capabilities within the app to fix and update the vulnerabilities present in the firmware. However, if the Bluetooth chip does not support OTA functionality, manufacturers need to take other measures to mitigate the impact of vulnerabilities and protect user safety:

1) Device Recall. If feasible, manufacturers should consider recalling hardware that cannot be upgraded. By retrieving affected devices, manufacturers can reissue them with updated secure firmware before reselling, ensuring that users receive fixed versions.

2) Provide Risk Warnings. If the recall is impossible, manufacturers should provide detailed vulnerability descriptions and potential risks to users, allowing them to understand potential security threats and take necessary preventive measures.

3) Withdraw Old Versions of the App. Cease the use of old versions of the app available on the Internet to prevent users from using software versions containing vulnerabilities.

Limitations. Limitations of this paper include the following aspects: (1) Unavailability of old app versions: although Androzoo contains a considerable number of apps, there is still a possibility that the old version of a particular app may not be found. (2) Implementation of code protection in old versions: If code protection exists in older versions, it can also make analysis more difficult. (3) Manual analysis: Code protection techniques were identified through manual analysis, which could potentially overlook certain code protection techniques or lead to misjudgments.

VII. Related Work

This section reviews previous work on BLE security and IoT security. Zhang et al. [36] mentioned the security risks of outdated Android apps. Inspired by it, our work demonstrates the feasibility of analyzing firmware vulnerabilities of BLE devices based on their outdated companion apps. Also, a series of further investigations and measurements were conducted.

BLE Security. With the advancement of Bluetooth technology, an increasing number of security research efforts are focusing on BLE attacks and defense, including identity tracking attacks that leverage the static UUIDs [37], the MAC address [35] and the advertised packets [29]. To mitigate these attacks, Wu et al. [33] proposed BlueShield, utilizing the BLE device identity information carried by advertising packets to filter out malicious packets from an attacker. Fawaz et al. [26] proposed BLE-Guardian, which provides channel-level protection to allow only authorized peripherals to connect with the protected device. In other aspects of BLE security, Tschirschnitz et al. [30] revealed a design flaw in the inconsistent association model in the BLE pairing process. Taking advantage of this fact, the method confusion attack can easily achieve an MITM position of two BLE devices. Xu et al. [34] identifies design flaws in the Bluetooth protocol and its implementation on Android systems.

IoT Security. In terms of IoT security, Wang et al. [31] presents a platform to accelerate the discovery and analysis of vulnerable smart home IoT devices. Wen et al. [32] presents FirmXRay, a static binary analysis tool, designed to detect vulnerabilities in BLE firmware used in IoT devices. Chen et al. [23] presents IoTFuzzer, an automatic fuzzing framework that detects memory corruption vulnerabilities in IoT devices without requiring access to their firmware images.

VIII. Conclusion

In this paper, we demonstrate the feasibility of analyzing firmware vulnerabilities of BLE devices based on their outdated companion apps. The insight is that BLE devices usually lack firmware update capabilities. Additionally, we give a series of concrete attack cases to show the security risks existing in outdated companion apps. The measurement results also confirmed that the lack of firmware updates is a widespread phenomenon. To facilitate this analysis approach, we implemented a framework, BLESecurAssist, which takes an IoT app as input and retrieves its outdated version. Furthermore, by integrating other analysis tools, BLESecurAssist can further execute more targeted analyses.

Acknowledgements

We thank the anonymous reviewers for their insightful comments. This work was supported by Taishan Young Scholar Program of Shandong Province, China (No. tsqn202211001).

References

[1] Androguard. https://github.com/androguard/androguard.
[2] Bluetooth Specification Version 4.0. https://www.bluetooth.com/specifications/specs/core-specification-4-0/.
[3] Bluetooth Specification Version 4.2. https://www.bluetooth.com/specifications/specs/core-specification-4-2/.
[4] CC2540 – Bluetooth Low Energy wireless MCU with USB. https://www.ti.com/product/CC2540.
[5] CSR OTA Firmware Update. https://developer.qualcomm.com/forum/qdn-forums/hardware/bluetooth-connectivity-iot/csr101x-product-family/csr1010/software/34169.
[6] CSR OTA Firmware Update. https://developer.qualcomm.com/qfile/34081/csr102x_otau_overview.pdf.
[7] Cypress OTA Firmware Update. https://github.com/Infineon/airoc-connect-android.
[8] ESP OTA Firmware Update. https://github.com/espressif/esp-iot-solution/tree/master/examples/bluetooth/ble_ota.
[9] JADX. https://github.com/skylot/jadx.
[10] Nordic Secure DFU Protocol. https://github.com/NordicSemiconductor/Android-DFU-Library.
[11] NXP OTA Firmware Update. https://www.nxp.com/docs/en/user-guide/UM10993.pdf.
[12] Realtex OTA Firmware Update. https://www.realmcu.com/en/Home/DnlDocuments/b83e55e7-72d0-41c1-9c3e-7639918bd176?t=2.
[13] Renesas OTA Firmware Update. https://www.renesas.com/eu/en/document/apn/gattbrowser-android-smartphone-application-instruction-manual-rev102.
[14] Renesas OTA Firmware Update. https://www.renesas.com/jp/ja/document/apn/firmware-update-ble-radio-ota.
[15] Silicon OTA Firmware Update. https://www.silabs.com/documents/public/application-notes/AN984-Implementing-Over-the-Air-Firmware-Upgrade.pdf.
[16] STM OTA Firmware Update. https://github.com/STMicroelectronics/BlueSTSDK_Android.
[17] STM OTA Firmware Update. https://www.st.com/resource/en/application_note/an5247-overtheair-application-and-wireless-firmware-update-for-stm32wb-series-microcontrollers-stmicroelectronics.pdf.
[18] Telink OTA Firmware Update. https://github.com/Ai-Thinker-Open/Telink_825X_SDK.
[19] The SoCs support OTA updates. https://novelbits.io/how-to-choose-ble-module-for-your-project/.
[20] TI Android Module for Over Air Download. https://git.ti.com/cgit/simplelink-ble-oad-android/simplelink-ble-oad-android.
[21] Wandoujia. https://www.wandoujia.com/about.
[22] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Androzoo: Collecting millions of android apps for the research community," in *Proceedings of the 13th International Conference on Mining Software Repositories*, ser. MSR '16, 2016.
[23] J. Chen, W. Diao, Q. Zhao, C. Zuo, Z. Lin, X. Wang, W. C. Lau, M. Sun, R. Yang, and K. Zhang, "Iotfuzzer: Discovering memory corruptions in iot through app-based fuzzing," in *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*, 2018.
[24] S. Dong, M. Li, W. Diao, X. Liu, J. Liu, Z. Li, F. Xu, K. Chen, X. Wang, and K. Zhang, "Understanding android obfuscation techniques: A large-scale investigation in the wild," in *Security and Privacy in Communication Networks - 14th International Conference, SecureComm 2018, Singapore, August 8-10, 2018, Proceedings, Part I*, 2018.
[25] J. Du, F. Xu, C. Zhang, Z. Zhang, X. Liu, P. Ren, W. Diao, S. Guo, and K. Zhang, "Identifying the BLE misconfigurations of iot devices through companion mobile apps," in *19th Annual IEEE International Conference on Sensing, Communication, and Networking, SECON 2022, Stockholm, Sweden, September 20-23, 2022*. IEEE, 2022, pp. 343–351. [Online]. Available: https://doi.org/10.1109/SECON55815.2022.9918597
[26] K. Fawaz, K. Kim, and K. G. Shin, "Protecting Privacy of BLE Device Users," in *Proceedings of the 25th USENIX Security Symposium (USENIX-Sec), Austin, TX, USA, August 10-12, 2016*, 2016.
[27] X. Jin, S. Manandhar, K. Kafle, Z. Lin, and A. Nadkarni, "Understanding iot security from a market-scale perspective," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, 2022.
[28] D. M. Junior, L. Melo, H. Lu, M. d'Amorim, and A. Prakash, "A study of vulnerability analysis of popular smart devices through their companion apps," in *2019 IEEE Security and Privacy Workshops, SP Workshops 2019, San Francisco, CA, USA, May 19-23, 2019*. IEEE, 2019, pp. 181–186. [Online]. Available: https://doi.org/10.1109/SPW.2019.00042
[29] A. Korolova and V. Sharma, "Cross-App Tracking via Nearby Bluetooth Low Energy Devices," in *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy (CODASPY), Tempe, AZ, USA, March 19-21, 2018*, 2018.
[30] M. von Tschirschnitz, L. Peuckert, F. Franzen, and J. Grossklags, "Method Confusion Attack on Bluetooth Pairing," in *Proceedings of the 42nd IEEE Symposium on Security and Privacy (Oakland), San Francisco, CA, USA, 24-27 May 2021*, 2021.
[31] X. Wang, Y. Sun, S. Nanda, and X. Wang, "Looking from the mirror: Evaluating iot device security through mobile companion apps," in *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*, 2019.
[32] H. Wen, Z. Lin, and Y. Zhang, "FirmXRay: Detecting Bluetooth Link Layer Vulnerabilities From Bare-Metal Firmware," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS), Virtual Event, USA, November 9-13, 2020*, 2020.
[33] J. Wu, Y. Nan, V. Kumar, M. Payer, and D. Xu, "BlueShield: Detecting Spoofing Attacks in Bluetooth Low Energy Networks," in *Proceedings of the 23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID), San Sebastian, Spain, October 14-15, 2020*, 2020.
[34] F. Xu, W. Diao, Z. Li, J. Chen, and K. Zhang, "BadBluetooth: Breaking Android Security Mechanisms via Malicious Bluetooth Peripherals," in *Proceedings of the 26th Annual Network and Distributed System Security Symposium (NDSS), San Diego, California, USA, February 24-27, 2019*, 2019.
[35] Y. Zhang and Z. Lin, "When good becomes evil: Tracking bluetooth low energy devices via allowlist-based side channel and its countermeasure," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, H. Yin, A. Stavrou, C. Cremers, and E. Shi, Eds., 2022.
[36] Y. Zhang, J. Weng, J. Weng, L. Hou, A. Yang, M. Li, Y. Xiang, and R. H. Deng, "Looking back! using early versions of android apps as attack vectors," *IEEE Trans. Dependable Secur. Comput.*, vol. 18, no. 2, pp. 652–666, 2021.
[37] C. Zuo, H. Wen, Z. Lin, and Y. Zhang, "Automatic Fingerprinting of Vulnerable BLE IoT Devices with Static UUIDs from Mobile Apps," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS), London, UK, November 11-15, 2019*, 2019.