Hindawi
Security and Communication Networks
Volume 2023, Article ID 3353548, 11 pages
https://doi.org/10.1155/2023/3353548

Research Article

Deception Attacks against Android-Based Smart Bracelets

Weimin Gao[1,2] **Jun Xie**[2] **and Xinlong Li**[2]

[1]*School of Computer Science and Engineering, Central South University, Changsha 410083, China*
[2]*School of Computer Science and Engineering, Hunan Institution of Technology, Hengyang 421002, China*

Correspondence should be addressed to Jun Xie; 625836126@qq.com

Received 18 August 2022; Revised 30 January 2023; Accepted 9 March 2023; Published 15 April 2023

Academic Editor: Saed Alrabaee

The rapid development of the intelligent equipment industry has promoted the emergence of a series of emerging industries, effectively improving the technical level of society and people's quality of life. Due to the inherent characteristics of smart devices, smart devices face serious privacy disclosure risks. Therefore, we have studied the security problem of the widely used smart device smart bracelet in this paper. We have adopted the attack method from easy to difficult to obtain the reading and writing instructions and timing characteristics of the intelligent devices to attack and steal private information. Specifically, we first attack through the strategy of log analysis; if this method is unable to obtain effective information, then we further acquire sensitive information on the basis of hook technology; and if the method on the basis of hook technology is still unable to obtain relevant information, then we will further use reverse engineering to conduct reverse analysis on the app to obtain sensitive information. Second, we develop a fake app on the basis of sensitive information and use it as a bridge to attack intelligent devices. In order to verify the effectiveness of the method, we successfully attacked and stole information from three popular business intelligence bracelets of different brands on the basis of the proposed method. The first step is to develop a fake app on the basis of the identified vulnerabilities. This app can bypass the protection measures of confusion and forced pairing and resetting to cheat the smart bracelet and can successfully enable or disable the jitters function remotely to modify the time and to obtain the sensitive data of the smart bracelet owner. In our attack process, we do not need the cooperation of the owner of the smart bracelet, nor do we need the target smart bracelet to match with our app.

1. Introduction

Nowadays, smart devices have become an inseparable part of the society and provide convenient services for the people. Since the advent of wearable devices, the halo on the body has not faded. Some people even think that wearable devices will become a next-generation product that subverts tradition and creates the future. Yet a few years later, wearable devices still remain in the media with high exposure. Under the spotlight, it rarely appears in people's lives. Why cannot smart wearable devices fly?

First of all, there is currently no industry standard in China, resulting in uneven product levels including charging speed, industrial design, data acquisition and transmission, and sensor accuracy, and since there is no exact standard scale, so products ranging from a few thousand yuan to tens of yuan can call themselves wearable devices. Another reason the public is not interested in wearables is that most features are neither innovative nor practical. Taking a brand of the smart bracelet as an example, the step counting and sleep monitoring functions provided by it are useless to many people when there are only data but no service. According to data, 76.9% of churn users use wearable devices for less than 3 months [1].

In addition, the industry believes that the most important issue is safety. At this stage, smart bracelets and smart watches are still the mainstream of wearable devices, providing people with convenient services. The security of these devices has aroused the researchers' attention. Due to their own inherent characteristics, wearable devices, such as smart glasses and smart bracelets, face significant risks of privacy leakage. In particular, first, wearable devices usually have limited computational capacity and memory. In order to guarantee the battery life or power supply time, most functions operate intermittently but not continuously, which brings up the chance of malicious attacks. Second, most wearable devices use TinyOS or LiteOS, in which the system's security policies and algorithms cannot be sufficiently complicated. Third, smart wearables interact with Android or IOS applications (apps) or other devices through wireless communication, providing convenience for grabbing data and consequently leading to security concerns. Fourth, the majority of wearable devices connect with the Internet via the smartphone apps so as to synchronize and update data in the cloud, and any vulnerability inside may result in potential privacy unveiling.

Zoologists use smart wearable networks for animal physiological activity monitoring and control, medical institutions use smart wearables for patient monitoring and notification, and they are used in the military for environmental tracking and habitat monitoring, etc. [1]. Zoologists use smart wearable networks for animal physiological activity monitoring and control, medical institutions use smart wearable for patient monitoring and notification, and they are used in the military for environmental tracking and habitat monitoring, etc [2]. Embedded sensors on wearable devices can be utilized to capture the motion information of users and have the possibility of leaking their sensitive information [3]. Sensor data of smartwatches were also reported for leaking information about what the user is typing on a regular keyboard under the condition that the smartwatch is only on the left hand [4]. Later, a similar work was conducted which exploited the sensors in smartwatches to infer user's highly sensitive information by devising a data training-based system [5]. Wang et al. showed that the sensors embedded in wrist-worn wearable devices, such as smartwatches and fitness trackers, can be utilized to discriminate mm-level distances of the user's fine-grained hand movements during the key-entry activities [6]. Pan et al. showed that it is possible to recover password input with a Bluetooth mouse and an on-screen keyboard by capturing Bluetooth communication packets [7]. It was further reported in [8, 9] that adversaries can obtain the readings of sensors in wearable devices via sniffing Bluetooth communications and analyzing the data packets.

For the distributed security state estimation under unknown spoofing attacks [10], a neural network-based mechanism is put forward to approximate the unknown falsified innovations with the aim to mitigate the effects on the estimation performance [11]. The Internet of Things (IoT) is a paradigm that connects objects to the Internet as a whole and enables them to work together to achieve common objectives, such as innovative home automation. Potential attackers see the scattered and open IoT service structure as an appealing target for cyber-attacks. So, security cannot be dealt with independently [12]. It provides an inclusive analysis of intrusion detection on the basis of deep learning techniques followed by different intrusion detection systems [13, 14]. DeepGuard is proposed, which is a framework of privacy-preserving backdoor detection and identification in an outsourced cloud environment for multiparticipant computation.

Given the abovementioned pioneering works and lots of efforts from the industry to further strengthen the security of smart wearable devices, it still faces significant threats of privacy leakage due to their own inherent characteristics. Towards this end, we reinvestigate in this paper the security concerns of the smart bracelet and present Android-based deception attacks against popular commercial smart bracelets of three different brands. As far as we know, a similar work to ours is that in [15] which proposes a fake app-based attack on a commercial smart bracelet. Our major contributions are summarized as follows:

(i) First, it is noticed that one prerequisite of the attack in [16] is the successful pairing process between the attacking smartphone and the targeted smart band, while in our deception attacks, no cooperation from the smart bracelet owner is required, neither the pairing process between the targeted smart bracelets and our fake app is required. All the three different smart bracelets selected in our work adopt the latest protection technology of forcible pairing and resetting, where any forcible pairing between the device and the nonofficial app will directly result in resetting of the whole smart bracelet system.

(ii) Another major difference is that the attack scheme in [16] can only obtain user privacy information of the targeted smart bracelet, while by installing our fake app, as long as the targeted smart bracelet is located within the communication range, we are able to remotely activate/deactivate the shaking function, to adjust or modify time, and in addition to obtain the smart bracelet owner's sensitive data.

(iii) Finally, our Android-based deception attacks are developed after successfully identifying the common vulnerabilities of the current mainstream smart bracelets and can be efficiently conducted towards three different commercial smart bracelets, rather than one single type.

The rest of the paper is organized as follows: in Section 2, we demonstrate the design motivation. Section 3 introduces vulnerabilities, common attacks, and countermeasures of Android apps. Section 4 describes our detailed methodology for security analysis, including log analysis, hook technology, and Android reverse engineering. Extensive experimental results are provided in Section 5. Finally, we conclude the whole paper in Section 6.

2. Background and Motivation

While smart bracelets bring convenience to users' daily life, they also bring personal data security risks [17]. Smart bracelets and smartphones may become the entrance of malicious attacks by hackers in the communication process, encryption system, sensors, permission control, and Bluetooth signals. Once the smart bracelet is attacked, it will not only cause damage to the device itself but also cause other devices associated with the attacked device to be controlled by hackers, resulting in loss of control of user equipment and data leakage. In general, smart bracelets and smart devices are most vulnerable to hacker attacks in the following four aspects:

(1) The application program of the device
(2) The sensor of the device
(3) The Bluetooth module of the device
(4) Communication process: including communication from the smart bracelet or smart devices to the official app of the smartphone, communication from the official smartphone app to smart bracelets or smart devices, and communication from the official smartphone app to the cloud

According to the latest Cisco Visual Networking Index [18], the total number of smartphones is predicted to be beyond 50 percent of the global devices by 2021. The malware, i.e., any software used to disrupt a system, to gather sensitive information, to display unwanted advertisements, or to do other abnormal actions, will inevitably become a major threat to the privacy of wearable devices among various applications in all aspects of life. As shown in Figure 1, hackers usually apply reverse-engineering towards the official apps, find out the loopholes, and develop fake apps accordingly with functions similar to that of the official apps, thus establishing malicious attacks. In this paper, we focus on the security concerns caused by the interaction between smart bracelets and smartphone apps. Since Android is the most popular and pervasive mobile device operating system, so we focus on the Android platform.

From the perspective of privacy risk protection of smart devices, the security of smart devices themselves is relatively low. The wireless transmission medium of smart devices is exposed to the outside, the network topology changes frequently, and the quality of Bluetooth transmission is good and bad. It can be seen that the stability of smart device communication is extremely susceptible to the limitations of its own software and hardware structure, the influence of the surrounding physical environment, and human information interference [19]. When the transmission line of the smart device fails, the data stored in the smart device lose its protection and it is prone to data theft, tampering, or loss. Some attackers will choose to mix other false and messy data in the data transmission process of smart devices to artificially interfere with the data transmission process of smart devices, thereby stealing or tampering with user data, affecting user experience and threatening data security [20].

Therefore, the research studies on the security of smart devices have an important practical significance and commercial value. In this paper, from the perspective of attackers, we study the security issues of communication between smart bracelets and mobile devices on the basis of Android operating systems.

3. Vulnerabilities, Attacks, and Countermeasures for Android Apps

The Android mobile phone system is an open-source mobile operating system, and its system composition mainly includes four parts: application system, kernel system, framework system, and system runtime library. Figure 2 shows common vulnerabilities, attacks, and countermeasures for Android applications. In the subsequent sections, we focus on the Android applications and discuss each of them separately.

3.1. App Vulnerabilities. According to a recent announcement by Altas VPN, it is understood that more than 60% of Android applications have numerous vulnerabilities, with an average of 39 dangerous vulnerabilities per Android application. These data are the results of Altas VPN's survey of 3,335 free and paid apps in the Google Play Store.

The vulnerabilities of Android apps generally can be classified into two categories: component exposure and sensitive application program interface (API) call. The component exposure of Android apps means that functions realized by the component can be maliciously called by the attackers or injected within the malicious data, thus affecting the normal operations of the application. The relaxed permission management of the sensitive API call also makes it possible for hackers to access the API and call it to facilitate malicious behaviors.

3.2. App Attacks

3.2.1. Repackaging App. Repackaging is a process of decompiling a popular app, inserting malicious code, recompiling the app, and distributing it to the app markets, which account for the majority of Android app's hacking cases. By using hacking tools and techniques, repackaging attacks can be successfully implemented against mission critical Android mobile applications [4]. The repackaged app with a malicious payload may send premium SMS messages stealthily, steal personal data, or purchase apps without the user's awareness [1, 21, 22]. This kind of hacking is mainly due to the fact that Android apps are usually written in Java language (although some have "native" C calls), and it is easy to reversely analyze the app with the existing reverse-engineering tools. There are many reverse-engineering tools, such as apktool and dex2jar.

3.2.2. Drive-By Download. A drive-by download is a kind of an attack that forces users to automatically download and install malware by redirecting them to malicious URLs. A drive-by download usually exploits an app that is out of date or has a security flaw. Since the job of the downloaded code is only to contact another computer where it can pull down the rest of the code on the smartphone, it is often very small so that the user probably would not notice it. For example, a file-sharing program might include a spyware program that tracks and reports the user information for targeted marketing purposes. An associated adware program can then generate pop-up advertisements using that information.

3.2.3. Dynamic Payload. The malicious payload can be embedded into an app as an executable apk/jar. The app will decrypt the payload once it is installed. Usually, the embedded APK disguises itself as an important update so as to coax the user to install it.

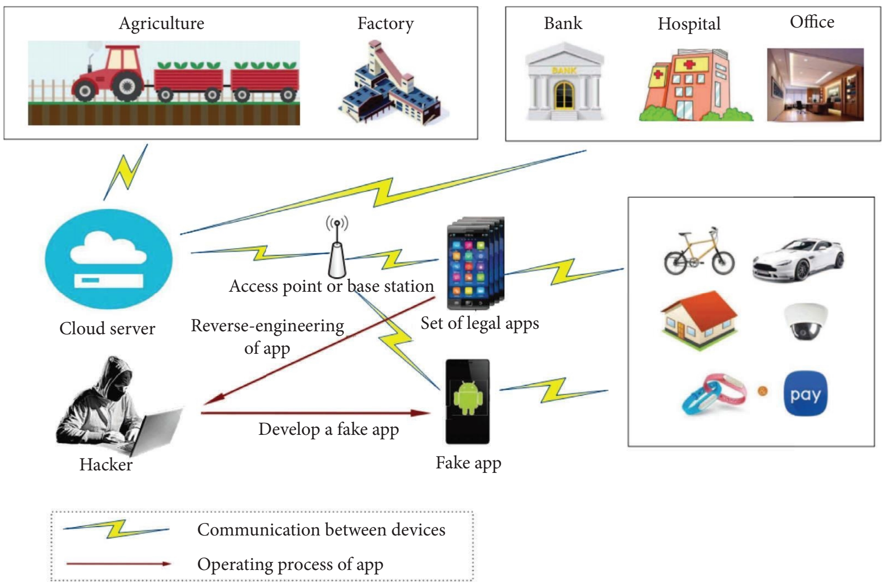


FIGURE 1: Illustration of Android-based attacks.

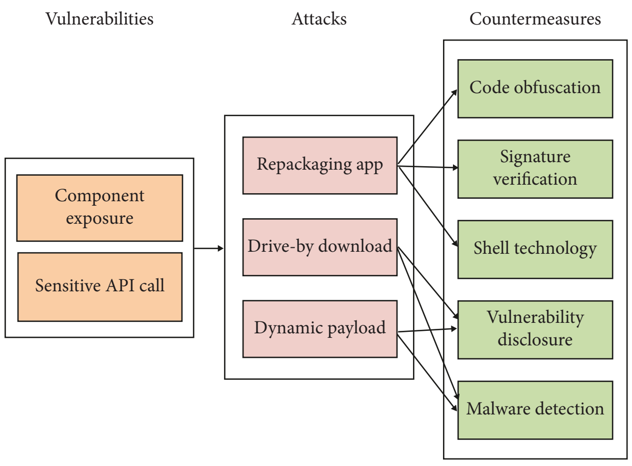


FIGURE 2: Vulnerabilities, common attacks, and countermeasures of Android apps.

3.3. Countermeasures

3.3.1. Code Obfuscation. Code obfuscation techniques transform a program so that it is difficult to understand while its functionality is identical to that of the original. However, code obfuscation only increases the difficulty in reading the source code, but it cannot play an efficient part in defense. Code obfuscation usually can be divided into three kinds: layout obfuscation, data obfuscation, and control obfuscation. Layout obfuscation alters the information unnecessary to the execution of the program, such as identifier names and comments. Data obfuscation changes the storage, the organization structure, and the order of the data in a program. Control obfuscation disguises the real control flow in a program.

3.3.2. Signature Verification. It is necessary to sign the new APK generated from the modified byte-code file before redistributing it to the app store. When installing an APK, the Android system will verify its digital signature information and check its integrity according to this. If the signature of an APK is different from the original APK, then the software is tampered. Thus, the system will stop running.

3.3.3. Shell Technology. Shell protection is a kind of code encryption technology. In fact, shell technology means using special algorithms to compress the resources of the executable file. The compressed file can run independently, and its decompression process is hidden completely in the memory. The shell program is executed before the execution of the original program so that it can obtain the control right and decrypt as well as restore the program. After the program is being restored, the shell program will return the control right to the original program and the original code will be executed. Adding a shell to the protected software makes it difficult for the software to be cracked. Shell technology and code obfuscation mentioned previously are the most popular techniques applied in antidecompilation.

3.3.4. Vulnerability Disclosure. The vulnerabilities of Android apps not only threaten the stability of the running process and security of the privacy information but also threaten the security of the whole system because of exposing the critical function. The vulnerability disclosure techniques aim to discover the potential vulnerabilities of the Android app so as to restore the vulnerabilities and protect the operation of the application. According to the difference in analyzed objects, the techniques of vulnerability disclosure can be classified as source code-based techniques and target code-based techniques.

3.3.5. Malware Detection. Malware detection can find out those apps with a malicious code. Usually, the malware detection techniques are divided into two kinds: static detection techniques and dynamic detection techniques. Static detection techniques analyze the code without actually running it; hence, their execution speeds are quick, and this method is simple as well as efficient. Dynamic detection techniques monitor the executed code and inspect its interaction with the system by extracting the critical data of the app operation process as a characteristic. Malware detection plays an important role in the protection of Android apps. Hu et al.propose a new system named MIGDroid that leverages the method invocation graph-based static analysis to detect repackaged Android apps [4].

4. Empirical Study on Commercial Smart Bracelets of Three Different Brands

4.1. General Protocol Stack for Smart Bracelets. The structure of the general communication protocol stack for smart bracelets is shown in Figure 3. Our research study concentrates on the UART profile layer, which lay above the BLE stack. The UART profile is realized at the smart bracelet end. There are two kinds of characteristics in the UART profile layer, one is the write characteristic and the other is the read characteristic. Every time the smartphone wants to interact with the smart bracelet, it first sends an operation code to the receiving interface of the smart bracelet through the write characteristic. If the smart bracelet can parse the operation code, then it will execute the corresponding operations. Finally, a value will be returned to the smartphone through the read characteristic of the smart bracelet to notify the smartphone whether the operation is executed successfully or not. Since we only implement our attacks on the UART profile layer, we can bypass the authentication between the smart bracelet and the smartphone as well as the operations related to the cloud server.

4.2. Methodologies. Through the description of the smart bracelet protocol stack, it is easy to know that the key point of the deception attack is to obtain the interactive instructions (write, read, and notify instructions) between the smart bracelet and the smartphone. Only when a specific instruction is written to the relevant characteristic can the attacker obtain the sensitive data contained in the smart bracelet or control the smart bracelet. There are three methods we use to obtain the instructions: log analysis, hook technology, and Android reverse engineering.

4.2.1. Log Analysis. We first find the process of the official app installed on the smartphone using a specific software tool and then dynamically receive the corresponding log files through the process. By analyzing the log files, we can obtain the instructions sent from the app to the smart bracelet and the returned results sent from the smart bracelet to the app. Figure 4 shows the instruction we obtained through this method. However, for some smart bracelets, the log files of their official apps are processed with security precautions; hence, the method fails in this situation.

4.2.2. Hook Technology. The hook is a message-processing mechanism as well as a program segment to process the message. The hook mechanism allows the application to the captured message and to get its control rights before the message is being transmitted to the target address. The specific Xposed module application installed on the smartphone can monitor the instructions of the interaction between the official app and the smart bracelet, as shown in Figure 4. However, the instructions of some smart bracelets have a timestamp, namely, the instruction sent to implement the same function every time is different. So, this method is not applicable to these smart bracelets.

4.2.3. Android Reverse Engineering. The process of applying shell removal, decompilation, program understanding, and other computer technologies to an executable app, then analyzing the structure, flow, algorithms, and code of the program, and finally inferring the app's source code and design principle, is called Android reverse engineering. Applying reverse engineering to an official app corresponding to a smart bracelet can help the attacker understand their interaction process and obtain the instructions. Reverse engineering is generally time-consuming and difficult. When the log analysis and hook technology both fail, reverse engineering can remedy this flaw. Some details about reverse engineering will be involved in the following contents.

4.3. Deception Attacks Based on Reverse Engineering. Since Android reverse engineering is complicated, it is necessary to introduce the process in detail. In the following, we first describe the overall attack idea, then the attack process is depicted, and finally, we illustrate the attack route. The Android reverse engineering process of the official app of the smart bracelet is shown in Figure 4.

4.3.1. Overall Attack Procedure

(i) We should be familiar with the interfaces, operations, and functions of the official app that matched with the smart bracelet. This step provides convenience for the analysis of the decompiled code.

(ii) We should get the targeted APK and check the type and the version number of its shell by using Android Killer. Then, we should remove its shell with the proper program and software tools.

(iii) We should decompile the APK file whose shell has been removed and analyze the decompiled code, locating the function we want to emulate.

(iv) We should develop an app according to the parameters and instructions found in the previous step, which can read data or modify some settings of the smart bracelet.

4.3.2. Detailed Attacking Procedures. Figure 5 shows the process of cracking an app, of which the most challenging thing is to apply reverse engineering to the app. In the

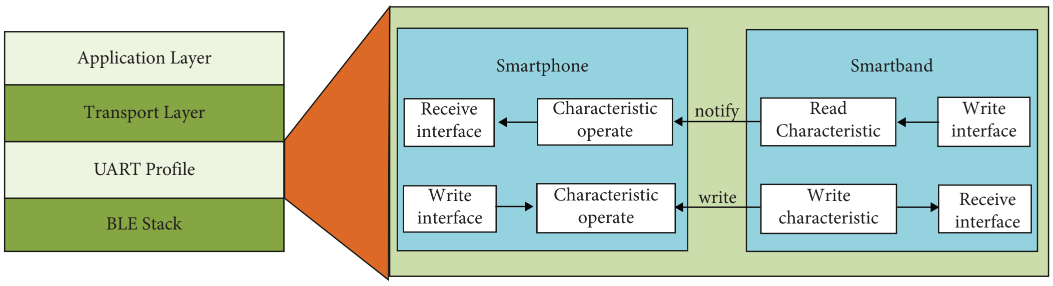


Figure 3: Communication protocol stack of the smart bracelet.

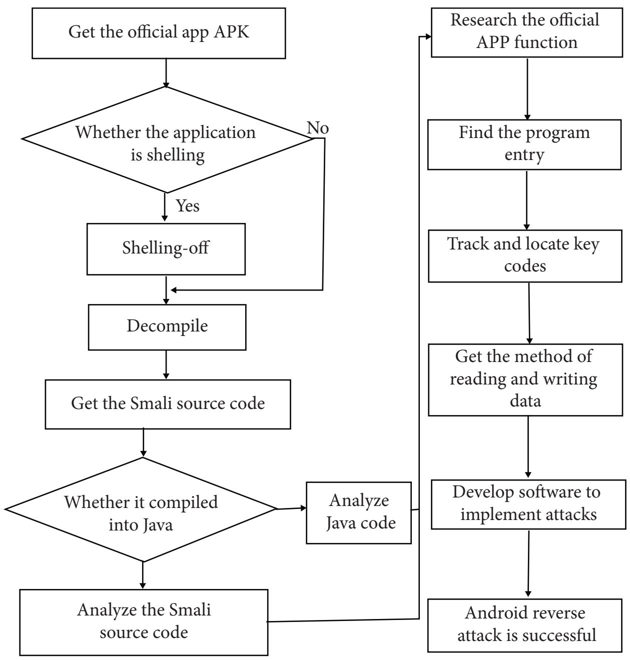

Get the official app APK
Whether the application is shelling
No
Yes
Shelling-off
Decompile
Get the Smali source code
Whether it compiled into Java
Analyze Java code
Analyze the Smali source code
Research the official APP function
Find the program entry
Track and locate key codes
Get the method of reading and writing data
Develop software to implement attacks
Android reverse attack is successful

Figure 4: The Android's reverse-engineering process.

reverse engineering stage, the first step is to remove the shell from the APK file. Since we do not need to repackage the app, we can bypass the signature verification. Then, we make an analysis of the code derived from the decompilation of the APK file. After reverse engineering, we develop a fake app in light of the instructions or methods found in the decompiled code. In the following paragraph, some details of the attack process are introduced.

(i) Shell Removal. We use Android Killer to check the shell type of the APK, and then we remove the shell manually using IDA Pro. The key idea of this procedure is to set a breakpoint on the dvmDexFileOpenPartial() function and dump the dex file in the memory.

(ii) Decompiled Code Analysis. After the abovementioned step, we begin to analyze the logic of the code. The analyzing process is shown in Figure 4. First, we make sure to get familiar with the interfaces of the official app and understand its functions along with operations. Second, we find the program entry of the project. In our experiment, the program entry is a welcome activity. Third, we track the code according to the implementation of the function we are interested in. Fourth, we locate the targeted function and finally find out the parameters and instructions which are necessary for the function to be realized. Table 1 shows the information on instructions we found in the decompiled code, and the instructions are represented by byte arrays. When the app sends an instruction to the smart bracelet, it will receive the returned value later, according to which the app can obtain the data contained in the smart bracelet or the results indicating whether the instruction is executed successfully or not. A challenge in this process is that there are many errors in the decompiled code resulting from the security reinforcement techniques of the app. Apparently, we cannot trust the code completely and need to avoid code traps.

4.4. Fake App Development.

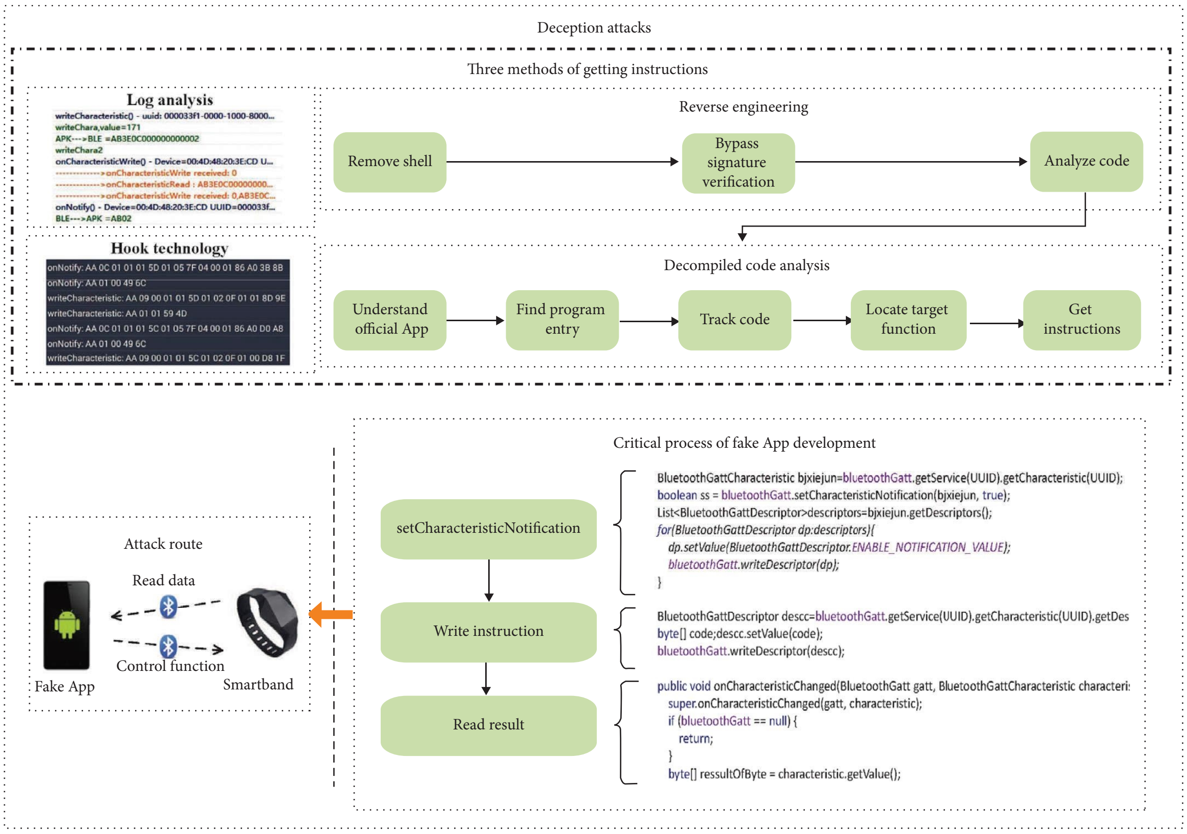
We first describe the function design of the fake app simply. According to the function and operating process of the official app, we design the function framework and the call relations among modules. Since the main function of the fake app is to obtain sensitive/health data and control the smart bracelet without authentication, the corresponding functional module should weaken the authentication process so that we can get the sensitive/health information and control the smart bracelet more directly. Hence, compared with the official app, the functions and interfaces of the fake app are simpler [10, 24–26]. The following are the concrete steps of the development process, and critical procedures together with the related code are shown in Figure 5:

Step 1. Discover nearby Bluetooth devices

The most basic functionality this fake app should implement is to discover the nearby Bluetooth devices which have not got paired with any other Bluetooth devices and add them to the list. This step is executed when the user taps the scan button. The Android system provides a BluetoothAdapter class, of which the bluetoothAdapter.startLeScan(callback) method can be used to scan peripheral Bluetooth devices. After the scanning procedure, the device information will be placed in the deviceList.

Step 2. Connect the smart bracelet to the smartphone

Bluetooth connections operate like any other communicating connections. There is a server and a client, which communicate via RFCOMM sockets. On



```
writeCharacteristic() - uuid: 000033f1-0000-1000-8000...
writeChara,value=171
APK--->BLE =AB3E0C000000000002
writeChara2
onCharacteristicWrite() - Device=00:4D:48:20:3E:CD U...
------------->onCharacteristicWrite received: 0
------------->onCharacteristicRead : AB3E0C00000000...
------------->onCharacteristicWrite received: 0,AB3E0C...
onNotify() - Device=00:4D:48:20:3E:CD UUID=000033f...
BLE--->APK =AB02
```

```
onNotify: AA 0C 01 01 01 5D 01 05 7F 04 00 01 86 A0 3B 8B
onNotify: AA 01 00 49 6C
writeCharacteristic: AA 09 00 01 01 5D 01 02 0F 01 01 8D 9E
writeCharacteristic: AA 01 01 59 4D
onNotify: AA 0C 01 01 01 5C 01 05 7F 04 00 01 86 A0 D0 A8
onNotify: AA 01 00 49 6C
writeCharacteristic: AA 09 00 01 01 5C 01 02 0F 01 00 D8 1F
```

```
BluetoothGattCharacteristic bjxiejun=bluetoothGatt.getService(UUID).getCharacteristic(UUID);
boolean ss = bluetoothGatt.setCharacteristicNotification(bjxiejun, true);
List<BluetoothGattDescriptor>descriptors=bjxiejun.getDescriptors();
for(BluetoothGattDescriptor dp:descriptors){
   dp.setValue(BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE);
   bluetoothGatt.writeDescriptor(dp);
}
```

```
BluetoothGattDescriptor descc=bluetoothGatt.getService(UUID).getCharacteristic(UUID).getDes
byte[] code;descc.setValue(code);
bluetoothGatt.writeDescriptor(descc);
```

```
public void onCharacteristicChanged(BluetoothGatt gatt, BluetoothGattCharacteristic characteri
   super.onCharacteristicChanged(gatt, characteristic);
   if (bluetoothGatt == null) {
      return;
   }
   byte[] ressultOfByte = characteristic.getValue();
```

Figure 5: Illustration of deception attacks, where the focus is on the process of reverse engineering of the official app and the development of the fake app. The fake app aims to read the data and modify some settings of the smart bracelet.

Android, RFCOMM sockets are represented as a BluetoothSocket object. Fortunately, most of the technical code for servers is handled by the Android SDK and available through the Bluetooth API. We obtain the RFCOMM socket from the desired BluetoothDevice by calling the method createRfcommSocketToServiceRecord(), using a 128-bit UUID which is similar to a port number.

Step 3. Open the setCharacteristicNotifification of the related characteristic

If the value of the BluetoothGatt.GATT SUCCESS is true, then we use BluetoothGATT to set the value of the related characteristic A to true in the method onServicesDiscovered().

Step 4. Write the instruction to the related characteristic

We write the instruction to the related characteristic B in the method onServicesDiscovered(), triggering the method onCharacteristicChanged().

Step 5. Read the returned results through the characteristic

The characteristic A gives the result to the app by notification while the result is the executed result of the instruction written to the characteristic B. At the same time, the result is a condition of whether to execute the next instruction.

5. Experiments and Results

5.1. Tools and Environment. Our experiment setup consists of five devices, three commercial smart bracelets, a smartphone, and a computer. In this paper, the brands of the smart bracelets are made anonymous and we only introduce the partial parameters of their hardware. Their MCU adopts W25Q80BV, STM32L151CBU6, and nRF51822 chips, respectively. The brand of the smartphone is Leno, and its model is K30-T; besides, its Android version is 4.4.4. The computer's OS is Windows 7. The software tool for shell removal is IDA pro v6.6. In order to reverse engineer on the official APK file, we apply Android Killer v1.3.1.0 to it. Log analysis also uses Android Killer v1.3.1.0. The fake app development uses Android Studio v2.2. Table 2 shows the information about experimental tools and the environment.

5.2. Experimental Results. We adopt log analysis to conduct research studies on the first smart bracelet. According to the instructions we find, we can obtain the motion data of the user from the smart bracelet. The results are shown in Figure 6.

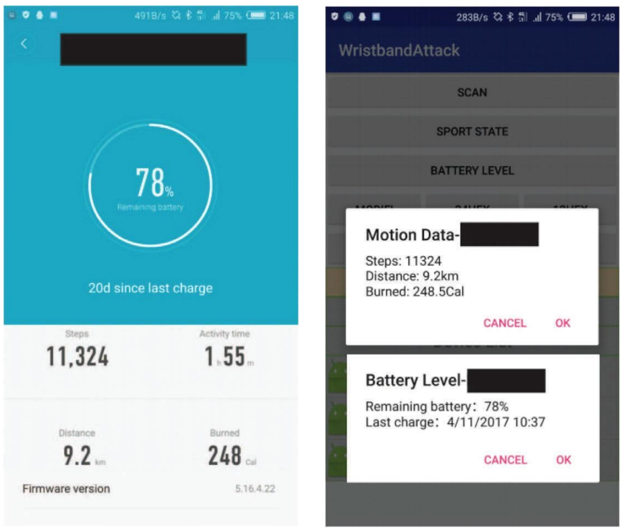
For the second smart bracelet, we used hook technology to find its instructions. Figure 7(a) shows that the fake app can get motion data (step number, distance, and calories burned) contained in the smart bracelet, and Figure 7(b) shows that the fake app can set an alarm for the smart

Table 1: Instructions found in the decompiled code.

Instructions	Function	Returned value type	Remarks
110, 1, 15, 1, −113	Read battery power	Byte array	The 3rd digit of returned byte array multiplied by 5 is current battery power
110, 1, 21, m, d, y, h, m, s	Set date and time	0 or 1	m, d, and y represent date; h, m, and s represent time
110, 1, 4, 1, −113	Read motion data	Byte array	The 12th and the 18th digits of the returned byte array represent the calories burned and the step number
. . .	-	-	-

Table 2: Experimental tools and environments.

Experiment equipment or software tool	Model or configuration
Smart bracelet1	MCU: W25Q80BV
Smart bracelet2	MCU: STM32L151CBU6
Smart bracelet3	MCU: nRF51822
Smartphone	Android 4.4.4
Computer	Windows7
IDA pro	v6.6
Android Killer	v1.3.1.0
Android Studio	v2.2

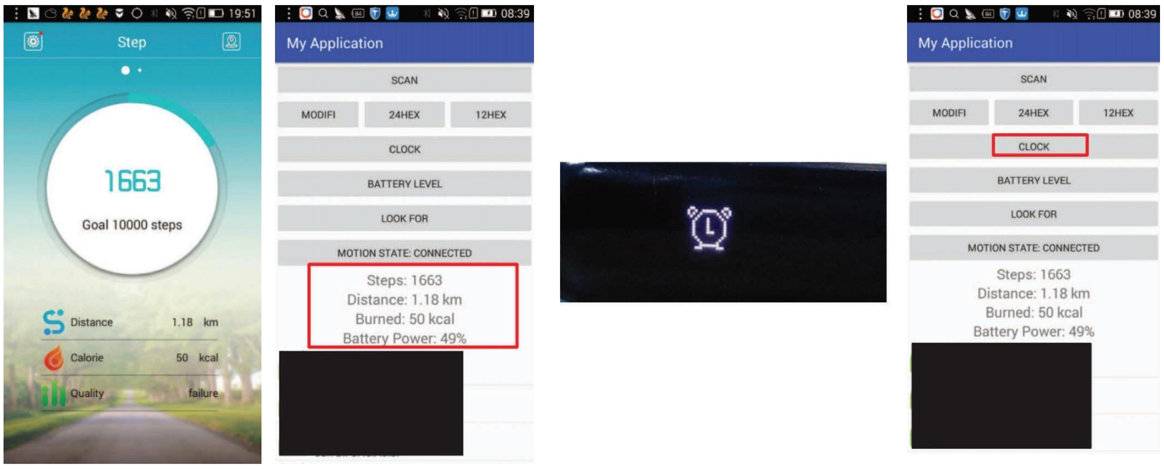


Motion Data-[redacted]
Steps: 11324
Distance: 9.2km
Burned: 248.5Cal
CANCEL OK

Battery Level-[redacted]
Remaining battery: 78%
Last charge: 4/11/2017 10:37
CANCEL OK

(a) (b)

Figure 6: Fake app (a) shows the same motion data as that of the official app (b).



Steps: 1663
Distance: 1.18 km
Burned: 50 kcal
Battery Power: 49%
[redacted]

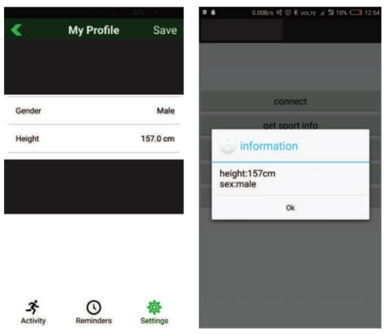
Steps: 1663
Distance: 1.18 km
Burned: 50 kcal
Battery Power: 49%
[redacted]

(a) (b)

Figure 7: Fake app (right) shows the same motion data as the official app (left) does and can set an alarm for the smart band. (a) Fake app (right) shows the same motion data as the official app (left). (b) Fake app (right) can set an alarm for the smart bracelet.

bracelet. Besides, the fake app can shake the smart bracelet so as to realize the search function. Unfortunately, the search function cannot be shown in the figures.

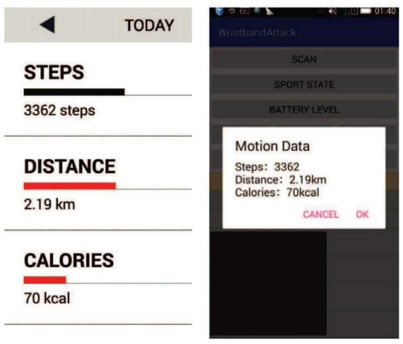
We apply reverse engineering to the third smart bracelet. From Figure 8, we can see that the fake app obtains the same data from the smart bracelet as the official app does.



Gender Male
Height 157.0 cm

height:157cm
sex:male

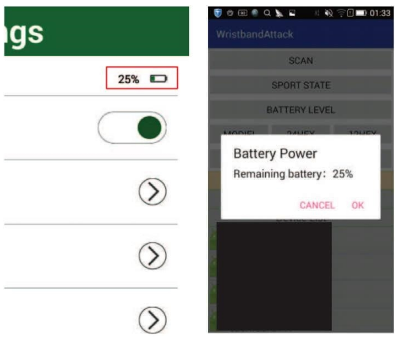
(a)



STEPS 3362 steps
DISTANCE 2.19 km
CALORIES 70 kcal

Motion Data
Steps: 3362
Distance: 2.19km
Calories: 70kcal

(b)

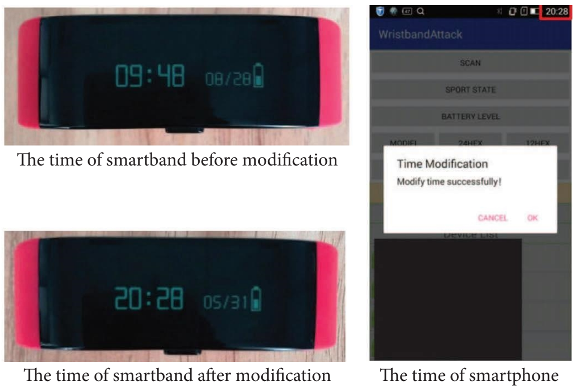


25%

Battery Power
Remaining battery: 25%

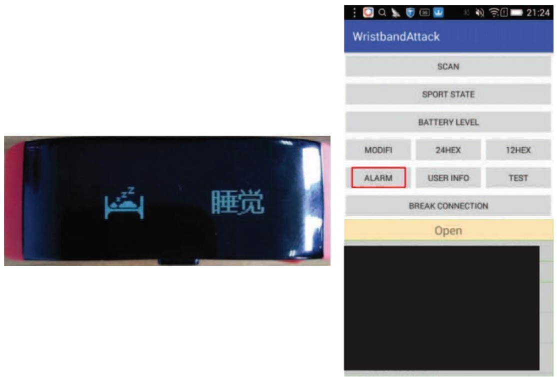
(c)

FIGURE 8: Fake app (right) shows the same privacy information, motion data, and battery power as the official app (left). (a) Privacy information. (b) Motion data. (c) Battery power.



Time Modification
Modify time successfully!

(a)



(b)

FIGURE 9: Fake app synchronizes the local time of the smartphone to the smart bracelet and sets an alarm for the smart bracelet. (a) Fake app synchronizes the local time of the smartphone to the smart bracelet. (b) Fake app sets an alarm for the smart bracelet.

Figures 8(a)–8(c), respectively, show that the fake app can get the privacy information (gender and height) of the user, motion data (step number, distance, and calories burned), and battery power from the smart bracelet. Figure 9(a) shows that the fake app can synchronize the local time of the smartphone to the smart bracelet, and Figure 9(b) shows that the fake app can set an alarm for the smart bracelet.

Our attacks work on the smart bracelets effectively. The deception attacks in this paper result from the vulnerability that the smart bracelet and smartphone do not authenticate each other at every connection time so that the smart bracelet cannot differentiate a legal user's smartphone from a hacker's one.

6. Conclusion

In this paper, we study the loopholes of three popular brands of business intelligence bracelets. We introduced the general methods of security analysis, mainly by forging an app and attacking a small number of intelligent devices in close range through the app. In addition, smart devices usually connect and communicate with the cloud through the app and penetrate the cloud. Through Android reverse engineering and the discovered vulnerabilities, a fake Android application was developed, which successfully spoofed the target three smart bracelets. The tested experiment results show that our fake application can obtain the sensitive data of the owner of the smart bracelet and remotely activate/deactivate some key functions of the smart bracelet. Of course, the attack is not the purpose. Through the attack, the defects and vulnerabilities of the device can be found so that the manufacturer can correct and improve it so that the intelligent device can serve people's lives more safely.

Data Availability

The data used to support the findings of the study can be obtained from the corresponding author upon request.

Disclosure

An earlier version of the manuscript has been presented as conference in "Cyberspace Safety and Security."

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant no. 61872387), the Philosophy and Social Science Foundation of Hunan Province (Grant no. 21YBA224), and the Hengyang Science and Technology Innovation Project, China (Grant no. 202250045133).

References

[1] Data Bridge, "Global wearable device industry market analysis in 2019," 2019, https://bq.qianzhan.com/reports/detail/300/191211-a7603d1f.html.

[2] E. Miluzzo, A. Varshavsky, S. Balakrishnan, and R. R. Choudhury, "Tapprints: your finger taps have fingerprints," in *Proceedings of the ACM MobiSys*, pp. 323–336, Durham, NC, USA, June 2012.

[3] Y. Ren, Y. Chen, M. C. Chuah, and J. Yang, "User verification leveraging gait recognition for smartphone enabled mobile healthcare systems," *IEEE Transactions on Mobile Computing*, vol. 14, no. 9, pp. 1961–1974, 2015.

[4] H. Wang, T. T. T. Lai, and R. R. Choudhury, "Mole: motion leaks through smartwatch sensors," in *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, pp. 155–166, New Orleans LA, USA, October, 2015.

[5] X. Liu, Z. Zhou, W. Diao, Z. Li, and K. Zhang, "When good becomes evil: keystore inference with smartwatch," in *Proceedings of the ACM CCS*, pp. 1273–1285, Toronto Canada, October, 2015.

[6] C. Wang, X. Guo, Y. Wang, Y. Chen, and B. Liu, "Friend or foe? Your wearable devices reveal your personal PIN," in *Proceedings of the ACM ACM ASIA CCS*, pp. 189–200, New York, NY, USA, May, 2016.

[7] X. Pan, Z. Ling, A. Pingley, W. Yu, N. Zhang, and X. Fu, "How privacy leaks from bluetooth mouse?" in *Proceedings of the ACM ACM CCS*, pp. 1013–1015, London, UK, November, 2012.

[8] M. "B. Ryan, "With low energy comes low security," in *Proceedings of the USENIX WOOT*, pp. 4–10, Santa Clara CA USA, August, 2013.

[9] R. Hasan, "StreetBit: a bluetooth beacon-on the basis of personal safety application for distracted pedestrians," in *Proceedings of the 2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC)*, IEEE, Las Vegas, NV, USA, January, 2021.

[10] L. Ma, Z. Wang, Y. Chen, and X. Yi, "Probability-guaranteed distributed secure estimation for nonlinear systems over sensor networks under deception attacks on innovations," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 7, pp. 465–477, 2021.

[11] A. Heidari and M. A. Jabraeil Jamali, "Internet of Things intrusion detection systems: a comprehensive review and future directions," *Cluster Computing*, vol. 12, pp. 1–28, 2022.

[12] A. R. Khan, M. Kashif, R. H. Jhaveri, R. Raut, T. Saba, and S. A. Bahaj, "Deep learning for intrusion detection and security of Internet of things (IoT): current analysis, challenges, and possible solutions," *Security and Communication Networks*, vol. 2022, Article ID 4016073, 13 pages, 2022.

[13] Y. Chen-Duo, Y. M. Wu, and M. Xu, "Privacy-preserving average consensus control for multi-agent systems under deception attacks," *Acta Automatica Sinica*, vol. 48, pp. 1–12, 2022.

[14] C. Chen, L. Wei, L. Zhang, Y. Peng, and J. Ning, "DeepGuard: backdoor attack detection and identification schemes in privacy-preserving deep neural networks," *Security and Communication Networks*, vol. 2022, Article ID 2985308, 20 pages, 2022.

[15] H. Huang, S. Zhu, P. Liu, and D. Wu, "A framework for evaluating mobile app repackaging detection algorithms," in *Trust and Trustworthy Computing*, pp. 169–186, Springer, Berlin, Germany, 2013.

[16] M. Lee, K. Lee, J. Shim, S. Cho, and J. Choi, "Security threat on wearable services: empirical study using a commercial smartband," in *Proceedings of the ACM IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*, pp. 1–5, Yeosu, South Korea, October, 2016.

[17] D. Ding, Z. Tang, Y. Wang, and Z. Ji, "Secure synchronization of complex networks under deception attacks against vulnerable nodes," *Applied Mathematics and Computation*, vol. 399, pp. 126017–126399, 2021.

[18] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, no. 4, pp. 393–422, 2002.

[19] J. Xie, S. Wu, Y. Li, J. Guo, W. Sun, and J. Liu, "My smartphone knows your health data: exploiting android-on the basis of deception attacks against smartbands," *Cyberspace Safety and Security*, vol. 22, pp. 291–306, 2017.

[20] K. S. Kim and B. S. Song, "Accessibility analysis of android-on the basis of smart phones targeted at people with upper limb dysfunctions," *Journal of Rehabilitation Welfare Engineering & Assistive Technology*, vol. 10, no. 4, pp. 267–272, 2016.

[21] W. Hu, J. Tao, X. Ma, and W. Zhou, "MIGDroid: detecting APP repackaging android malware via method invocation graph," in *Proceedings of the ACM IEEE International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–7, Athens, Greece, March, 2014.

[22] X. Zheng, L. Pan, and E. Yilmaz, "Security analysis of modern mission critical android mobile applications," in *Proceedings of the Australasian Computer Science Week Multiconference*, New York, NY, USA, January, 2017.

[23] W. Hu, J. Tao, X. Ma, W. Zhou, S. Zhao, and T. Han, "MIGDroid: detecting APP-repackaging android malware via method invocation graph," in *Proceddings of the IEEE ICCCN*, pp. 1–7, Shanghai, China, August 2014.

[24] S. Khan, M. U. Farooq, and M. O. Beg, "BigData analysis of stack overflow for energy consumption of android framework," in *Proceedings of the ACM International Conference on Innovative Computing (ICIC)*, Lahore, Pakistan, January, 2019.

[25] H. C. Kang and J. W. Jwa, "Development of android on the basis of smart tourism application on the basis of on tourism bigdata analytics," *Journal of Engineering and Applied Sciences*, vol. 15, no. 3, pp. 1164–1169, 2018.

[26] K. Chen, P. Wang, Y. Lee et al., "Finding unknown malice in 10 seconds: mass vetting for new threats at the google-play scale," in *Proceedings of the 24th USENIX Conference on Security Symposium*, pp. 659–674, New York, NY, USA, August, 2015.