sustainability

MDPI

Article

Efficient Protection of Android Applications through User Authentication Using Peripheral Devices

Jinseong Kim and Im Y. Jung *

School of Electronics Engineering, Kyungpook National University, Daegu 41566, Korea; jin7733@knu.ac.kr or pooba0@daum.net

* Correspondence: iyjung@ee.knu.ac.kr; Tel.: +82-53-950-7237

Received: 22 March 2018; Accepted: 20 April 2018; Published: 22 April 2018

Abstract: Android applications store large amounts of sensitive information that may be exposed and exploited. To prevent this security risk, some applications such as Syrup and KakaoTalk use physical device values to authenticate or encrypt application data. However, by manipulating these physical device values, an attacker can circumvent the authentication by executing a Same Identifier Attack and obtain the same application privileges as the user. In our work, WhatsApp, KakaoTalk, Facebook, Amazon, and Syrup were subjected to the Same Identifier Attack, and it was found that an attacker could gain the same privileges as the user, in all five applications. To solve such a problem, we propose a technical scheme—User Authentication using Peripheral Devices. We applied the proposed scheme to a Nexus 5X smartphone running Android version 7.1 and confirmed that the average execution time was 0.005 s, which does not affect the other applications' execution significantly. We also describe the security aspects of the proposed scheme and its compatibility with the Android platform and other applications. The proposed scheme is practical and efficient in terms of resource usage; therefore, it will be useful for Android users to improve Android application security.

Keywords: Android vulnerability; Android protection; Same Identifier Attack; User Authentication using Peripheral Devices; Android security

1. Introduction

The number of Android application downloads is increasing every year [1]. Android applications, which are often called Android apps, offer various conveniences to users. However, they contain a number of security threats [2–9]. By inserting malicious code into an application and distributing it, it is possible to steal user information or recover messages, photographs, etc., sent and received by a user's social network service (SNS). Moreover, if an attacker gains access to an account or account token, he/she can use the user's account.

In addition to the security threats listed in Table 1, information can be acquired from backup files on the Android system. The original purpose of the backup files is to help restore the user's information. However, if an attacker misuses them, he/she can analyze the application data in the backup files to obtain personal information for applications such as WhatsApp or Facebook [10,11]. Another form of attack hacks a user's Android application using backup technology. This method uses the Android Debug Bridge (ADB) to acquire backup data from an unrooted terminal, and then modifies the XML values of the application and restores the backup file [12]. Another method inserts a malicious application into the backup file and restores it [13]. Several methods have been studied to protect application data by enhancing the security of databases [9,14–16]; however, most of these techniques use the International Mobile Equipment Identity (IMEI) value to generate a key to the database file, which can be decrypted by an attacker [17].

Table 1. Android vulnerabilities published in Common Vulnerabilities and Exposures (CVE) [18].

Year	DoS	Code Execution	Overflow	Memory Corruption	SQL Injection	Bypass	Gain Information	Gain Privileges
2014	2	4	1		1	1	2	2
2015	56	70	63	46		20	19	17
2016	104	73	92	38		48	99	250
2017	44	139	59	26		28	52	34
Total	217	294	219	113	1	102	176	309
%All	20.9	28.3	21.1	10.9	0.1	9.8	17	29.8

In addition to such attacks, there is a new attack called the Same Identifier Attack (SIA) [19,20] that can create a copy of an application with the same privileges, using the user's backup file.

In this paper, we introduce the SIA concept and its method of attack, and show the results for five applications subjected to SIA: WhatsApp [21], which is used worldwide; KakaoTalk [22,23], which is widely used in Korea; Facebook [22], a representative SNS application; Amazon, a typical global online shopping application; and Syrup [24], Korea's integrated membership management application. In addition, we discuss what an attacker can do, such as initiating real-time conversations, sending and receiving files, gaining activity logs, logging into other linked accounts, conducting order inquiries, and obtaining details regarding debit/credit card information. To prevent these, we propose a scheme, User Authentication using Peripheral Devices (UAPD), which authenticates the smartphone from a user's other devices such as those paired by Bluetooth or configured through WiFi, to defend against SIA. This is outlined in Figure 1. We use nonrooted smartphones as the target of attack and for implementing UAPD. UAPD is part of the Android Framework and is compatible with all applications. In addition, UAPD authenticates the user from preconfigured peripheral devices; therefore, the time of execution required for user authentication is much less. Through testing, we confirmed that the average execution time is 0.005 s.

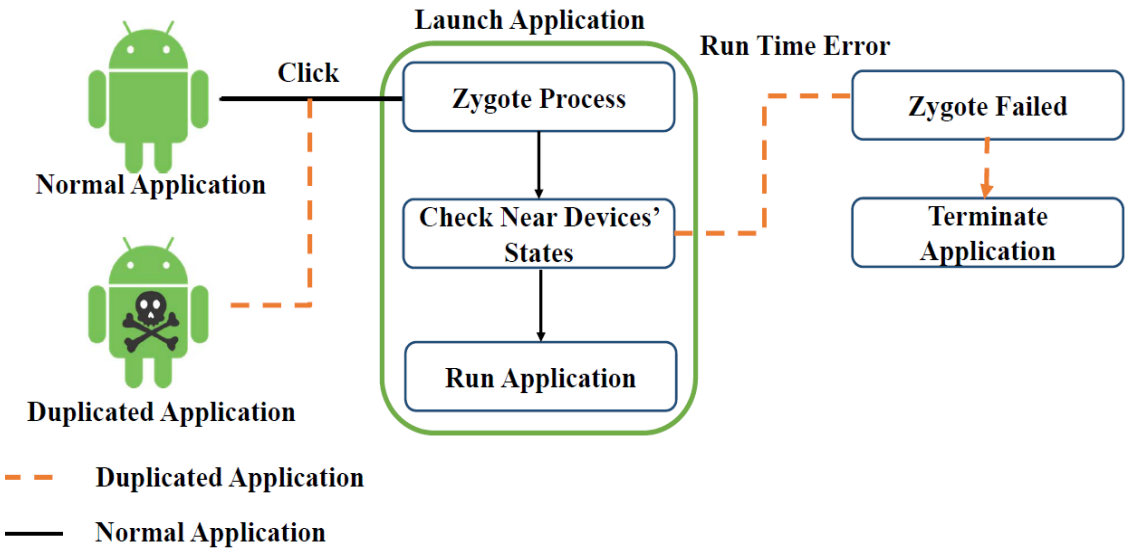

Figure 1. User Authentication using Peripheral Devices (UAPD) protecting Android applications against Same Identifier Attack (SIA).

The contributions of this paper are as follows.

- **Introduce an application duplication attack that can use the same user privileges**
  The creation of a same-user application without the user's knowledge can be exploited in a variety of ways. In particular, the attack target is an unrooted device. We also present the risk to Android, at present, because it is possible to create a copy of an application with the same user privileges by modifying the physical device values to duplicate the user's application.

- **Propose an improved defense of Android applications, which does not depend on the smartphone manufacturer**
  In this paper, a new scheme, UAPD, is proposed to defend Android applications, which does not depend on the smartphone manufacturer. UAPD authenticates the smartphone from a user's other devices such as paired Bluetooth or configured WiFi devices.
- **Avoid exposure to data replication attacks**
  Our proposed framework ensures that UAPD protects the user's application even if the attacker modifies the physical device values, because it does not use the IMEI, International Mobile Subscriber Identity (IMSI), or phone numbers, which can be modified.
- **Optimize performance for real applications**
  UAPD authenticates the user from a device that is preconnected to the smartphone. Therefore, it does not spend much time scanning for peripheral devices; the user authentication time is 0.005 s, on average.
- **Compatible with all Android applications**
  All Android applications go through the same processes for execution. We have developed UAPD to be compatible with all Android applications, by adding a user authentication process during its runtime setup.

The remainder of this paper is organized as follows. First, in the Related Works Section, we describe the SIA and other similar types of attacks. We also describe the studies that have been conducted to protect the information of users, for comparison with the method proposed in this paper. The SIA Section presents some background information to understand SIA and its attack methods. It also presents experimental results demonstrating the privileges acquired by executing SIA on WhatsApp, KakaoTalk, Facebook, Amazon, and Syrup, and discusses the risks of this attack. The next section describes the UAPD that can prevent SIA, the structure of UAPD, and its execution steps. The Evaluation Section presents the performance and overhead of UAPD application. We also present a scenario for UAPD, demonstrate that it can defend Android applications against SIA in this situation, and discuss its security stability. Finally, the Conclusion summarizes the previous sections and provides a description of the future work.

2. Related Works

Android stores the information used by applications. Application data are mainly stored in an SQLite database. In communication applications, information such as the conversation history and user's ID are stored. In addition, the information necessary for games, such as the account and cache, are also saved. Therefore, if the application data can be analyzed, the information about its user can be obtained. Dai et al. [17] explained the string decryption algorithm used in WeChat and WhatsApp and the decryption key formula for the encrypted database file structure, and showed that the key generated could be decrypted because it used only the IMEI value. Jain et al. [25] used the SQLite database vulnerability of Android to classify the risks of applications such as WeChat and WhatsApp, and showed that it was possible to modify the database values. While these attacks require rooting an Android device, attacks that can manipulate an application without rooting have also been developed.

Hacking Android applications using backup techniques [12] is a method for restoring the XML values of a specific application and analyzing the backup data using Android's ADB without acquiring root privileges after acquisition. It has been shown that an attacker can change features in an application by using the modified XML. In addition, there exists an Android ADB backup Android Package (APK) injection vulnerability [13], wherein the data stream is not filtered when using the Android backup manager. If a malicious APK is injected into the backup file and the user restores the data from the backup file, the malicious APK will be installed. The passive content leaks and pollution in Android applications [26] can be used to gain access to information in applications, by using the default setting

of Android: exported = "true". In [27], it was shown that the vulnerabilities of the database could lead to communication between malicious and normal applications. These attacks are possible without rooting only if some application information can be obtained or modified.

SIA can be considered to be more critical than the existing attacks because an attacker can use the behavior of all applications available, similar to the original user. Therefore, if an attacker attacks a user using SIA, he/she can obtain application information and impersonate the original user through the application. This attack utilizes the user's backup file; even if the user's phone is not rooted, it is possible to copy the target application completely. Therefore, the amount of information or authority ultimately obtained by the attacker is much higher than that of the other existing attacks.

The existing methods used for the protection of application data mainly encrypt the SQLite database. Mutti et al. [14] proposed SeSQLite, which integrated SELinux access control functions with SQLite, to manage the granular access policy for database objects. As a result, the database security of Android was enhanced. In a similar fashion, Paraboaschi et al. [15] added AppPolicyModules to SeSQLite, allowing application developers to specify system-level protection for the resources from specific applications. They also suggested interprocess communication with SELinux to prevent other applications from accessing specific applications' databases and viewing information. However, this method is vulnerable to SIA because it is limited to protecting application data from unauthorized or suspicious applications. If the attacker changes to the same physical value as the user's smartphone, as in SIA, and runs the application, the device will be recognized as authorized and this method will be disabled.

Ardiansa [16] suggested a method for encryption and decryption when writing or reading an Android database. This method has been added to the Android Software Development Kit (SDK), so that developers can easily protect the applications' data. In addition, because of the encryption/decryption, if the key is not exposed, it will be strong enough to protect the data. However, we uncovered three problems with this method. First, not all developers use this method, as it is costly and hassling to establish a separate setup, and there may be no need to worry about security for a particular application. Therefore, many applications are still vulnerable to the attacks using application data, such as SIA. The second issue is key management. Most of the methods including the Ardiansa proposal are developed for unrooted smartphones. Therefore, if you manage the keys for encryption/decryption in the system area or lower-level areas, the keys may be secure in unrooted smart phones; however, when you back up Android, the backup file will contain all the data needed to run the application. Therefore, if the attacker uses the backup file to perform SIA, the data encrypted by the SQLCipher will be decrypted automatically. This ultimately leads to the SQLCipher being disabled by SIA.

The Same Identifier Attack Defensor (SIAD) was proposed to solve these problems comprehensively [19,20]. SIAD is an automated process for protecting against SIA. It identifies the user's device before the application starts, and it encrypts and decrypts the application data before and after the application runs. It also prevents data exposure during backup. Instead of using the information on the device, new elements are added to each application and are used to validate and defend against SIA. It also encrypts the database for each application to improve the security of the application. However, this proposal will take a long time to run and exit the application if the database size of the application is large. In addition, as it is designed to be compatible with all applications, it results in unnecessary encryption/decryption. Therefore, we improved this approach to defend against the existing SIA and focused on reducing the time overhead that could occur when applying the proposal.

We propose a solution, UAPD, to solve the aforementioned problems. The proposed scheme can protect Android applications against SIA even if an attacker sets up a physical value that is the same as that of the user's smartphone or acquires the application backup data. In addition, unlike SIAD, UAPD does not encrypt/decrypt the application data. Therefore, it is much faster than SIAD and can prevent SIA effectively.

3. Same Identifier Attack

This section introduces the SIA. We describe the attack procedure of SIA briefly, and show the results for five representative applications subjected to SIA.

SIA occurs because of the verification processes required by applications and the structure of backup data.

Applications using sensitive information on smartphones should be able to verify that the users are correct in their execution. However, many applications do not carry out this verification process. Some applications such as Syrup, WeChat, and KakaoTalk use Short Message Service (SMS) authentication or the IMEI or Universal Subscriber Identity Module (USIM) information. Performing identity verification using SMS each time is burdensome, and it is ambiguous to define certain steps to be performed only in the case of suspicion. In the case of verification procedures using physical device values such as the IMEI and USIM information, these values may be exposed along with the values of the stored data, as many applications use the right to obtain them. Alternatively, this verification process is vulnerable to attack because identifiers can be changed directly using an Android emulator.

The backup method in Android can be changed according to the backup method supported by the manufacturer, the backup method using ADB, and the method by the backup application. Nowadays, there is an option to protect the backup file using passwords. However, this is not a compulsory option. If implemented, it will be necessary to remember the password. Moreover, decryption is possible without knowing the password because the file can be copied, and there is no limit to the number of attempts at decryption. Therefore, if the backup file is not safe and can be acquired, SIA is possible.

In order to understand an SIA, the Android application data structure is described below. The Android file system is divided into six partitions, as shown in Figure 2. Among them, the data area located in '/data/data', called the application directory, contains information related to the Android applications. Permission to access the data area is configured by an Android policy [28]. Table 2 lists the subdirectories of the application directory. Depending on the type of application, a person can obtain information such as the application usage signs, login information, and user traces [8].

Figure 2. Android partitions.

Table 2. Application data subdirectories [29].

Sub Directory	Description
shared_prefs	XML file of shared preferences
lib	Custom library files required by application
files	Developer-saved files
cache	Files cached by application
databases	SQLite and journal files

Figure 3 illustrates the attack procedure of SIA. As a precondition, the attacker must obtain the backup data that are not encrypted by the user. It is quite easy to copy the backup file of any application stored on a smartphone. There are three ways for an attacker to retrieve the backup file from a remote location. The first is to retrieve the backup files ("ad", "bak", "Tibkp", etc.,) stored in the cloud (such as Google, Baidu, or OneDrive) or on an individual server, through the Internet. In this case, the attacker does not have to do anything with the Android phone because the process involves just a simple download. This means that the attacker need not acquire the root permissions for the user's smartphone for the attack. The second is to install a malicious app on the user's phone and instruct it to back up. The backup app is built into the Android phone, and therefore, rooted permissions are not required. The third is a physical approach. The attacker installs a malicious application that can search

for the backup files and send them to him/her. Alternatively, the attacker can physically connect a user's smartphone to a PC to acquire the backup files. Of course, the method involving an attacker's direct access to the device is not an effective attack method. However, since the consequences of the attacks can be critical, the effort to attack is intensified.

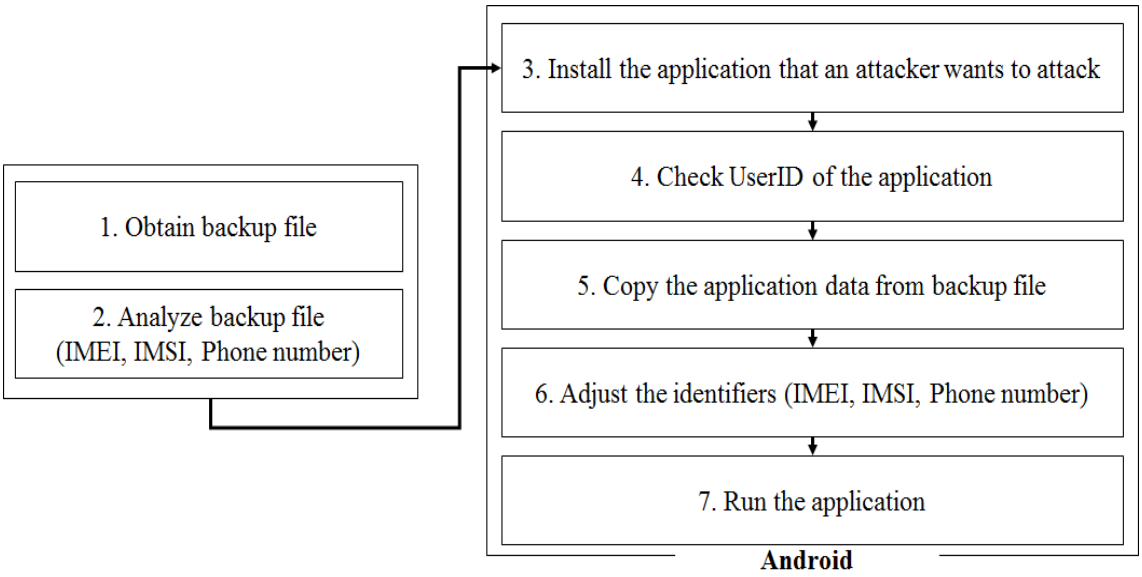

Figure 3. Same Identifier Attack [19]. IMEI, International Mobile Equipment Identity; IMSI, International Mobile Subscriber Identity.

After acquiring these data, the attack progresses through six stages. In the analysis step, the attacker selects a target application and parses information such as the IMEI, IMSI, and phone number in the backup file. Then, the attacker accesses the target application location ('/data/data/(App_package_name)') and checks the UserID and Group ID. Next, the target application data in the backup file are copied to the installed target application directory, the permissions of the copied folders and files are changed, and the IMEI, IMSI, and phone number are set as backup file information. Once this process is complete, the attacker can run an application with the same permissions as the user. In other words, the attacker has a replicated application with the same privileges as the user, with the user being unaware of this situation.

Table 3 shows the skills, tools, and background knowledge required by an attacker to perform the SIA. It also shows the attack difficulty level. When an attacker attacks with the SIA, the affected target device does not need root privileges. Once the attacker has the backup files, the attack can be carried out easily because not many tools or skills are needed.

Table 3. Same Identifier Attack requirements.

	Required Skill	Required Tool	Background	Difficulty
Acquiring backup files	Web searching Malicious application ADB	PC USB cable Extract tool	Depends on method	Depends on method
Analyzing backup files	Ability to analyze backup files	Analyze tool	Backup file structure	Easy
Creating copied application	Enter command	PC Smartphone	—	Easy

Table 4 lists the most popular applications in the Google Play Store across 103 countries. Table 5 lists the applications targeted for our attack. Messenger, in the communication category, and Instagram, in the social category, are targeted via WhatsApp and Facebook because they are linked with Facebook

accounts. Moreover, Korea's KakaoTalk and Syrup applications are considered targets because they can cause more damage, if exploited, than the previously mentioned applications. KakaoTalk is an instant messaging (IM) application, similar to WhatsApp, but it supports various other functions such as emoticons, gift cards, account transfer, and shopping. Syrup is a customized application in Korea that helps you to use and manage your memberships. Finally, Amazon, which is the top application in the shopping category, is targeted for representing applications in categories that could lead to secondary damage if abused.

Table 4. Play Store Ranks [30].

Name	Number of Countries	Category
Messenger	88	Communication
WhatsApp	69	Communication
Facebook	56	Social
Instagram	47	Social
Fidget Spinner	40	Arcade (game)
Viber	17	Communication
Wish	16	Shopping
Pokémon: Magikarp Jump	14	Simulation (game)
Snapchat	8	Social

Table 5. Applications targeted for SIA.

Name	Install Range	Category	Influence
WhatsApp	1 B–5 B	Communication	High
Facebook	1 B–5 B	Social	High
KakaoTalk	100 M–500 M	Communication	High
Amazon	100 M–500 M	Shopping	Middle
Syrup	10 M–50 M	Life Style	Middle

WhatsApp, KakaoTalk, Facebook, Amazon, and Syrup were installed on a Nexus 5X running Android versions 6.0 and 7.1, and were subjected to SIA that created a backup file. The backup file was created using ADB backup. If ADB backup was not available, as in the case of Facebook and WhatsApp, Huawei backup was used. The Android emulator used Android version 4.4 of the Nox App Player 3.8.1.2 [31]. In addition, root privileges were used to change the IMEI, IMSI, and phone numbers. After changing these values, the root privileges were revoked, and the applications were executed.

All five applications were attacked successfully. Table 6 summarizes the possible information leaked by the SIA. WhatsApp, KakaoTalk, and Facebook could be used to impersonate others, which could lead to secondary damage due to the abuse of social-engineering hacking. In addition, KakaoTalk and Facebook were linked with other applications or websites; therefore, SIA was more dangerous, as additional information could be collected from linked websites or applications. Unlike other applications, KakaoTalk allowed an attacker to use unused gift cards because the attacker had the purchase details and numbers and could collect information such as the user's address and private information by checking the user's payment history. Because Amazon could store the debit/credit card number used for payment, some card information could be leaked, and the attacker could change the shipping location or collect the user's address through the order history, similar to KakaoTalk. Finally, when Syrup used a membership card, the store name was displayed; thus, the radius of the user's movement could be ascertained and the user's location can be viewed any time. In addition, because membership card numbers were disclosed, membership points were easy to use, which could cause problems.

Table 6. Results of the SIAs.

Application	Chat History	Account Information	Account Links	Payment Information	Others	Version
WhatsApp	○	○			Receiving and sending files	2.17.146
KakaoTalk	○	○	○	○	Mobile voucher inquiry Receiving and sending files	6.2.0
Facebook	○	○	○		Security code generation	120.0.0.18.72
Amazon		○		○		5.1.0
Syrup				○	User's location	5.4.1

○ : Acquire information.

4. New Defense against SIA

This section describes the User Authentication using Peripheral Devices (UAPD) scheme, as a defense against SIA. UAPD is a method that authenticates the user's smartphone by using the user's devices such as smart watches, smart bands, Bluetooth earphones, WiFi routers, etc., which are usually used by the original user. We added some routines to the Zygote process. We first describe the background for understanding UAPD, and then we describe the overall process of UAPD.

4.1. Background

The Zygote process has been used from the initial version of Android to the present [32]. When an application is run on an Android system, Zygote is called. It is a neutral process that is forked ahead of time in order to run applications quickly on Android systems.

In Android, if you click on an application to run it, startActivity function will be called from the Launcher, and the Activity Manager will be called, as shown in Figure 4. This will initialize the Dalvik VM from Zygote using startVizZygote function, load the resources in advance, and create a child process using fork function. In order to run an application, the child application executes the application process by passing the child process from the Zygote to the execution flow of the application class. When all applications are executed, the Zygote process is performed. Therefore, when the Zygote process is modified, the running application will follow the operation of the modified Zygote process. UAPD defends against SIAs by modifying the Zygote and the application end routine of the Android runtime area shown in Figure 5.

There are multiple ways to terminate Android applications, e.g., finishAffinity function, finishAndRemoveTask function, and killProcess function. For all application termination methods excluding killProcess function, an application is terminated by the Destroy function function. For finishAffinity function, all parent activities can be closed in any activity, and since there is no need to use finish function in a root activity, the current activity does not have to go to the root activity. For finishAndRemoveTask function, the activity is quit, and the application is removed from the Task Manager. However, even though the application is terminated, the process is not terminated. Therefore, it is possible to confirm that the application class's onCreate function will not be executed if the application finishes termination by calling finishAndRemoveTask function and then runs again. In the case of killProcess function, the Destroy function should not be called. This is a command for terminating a process. If one activity is running, it is completely terminated. However, if several activities are running, the process is terminated, and the previous activity is executed again.

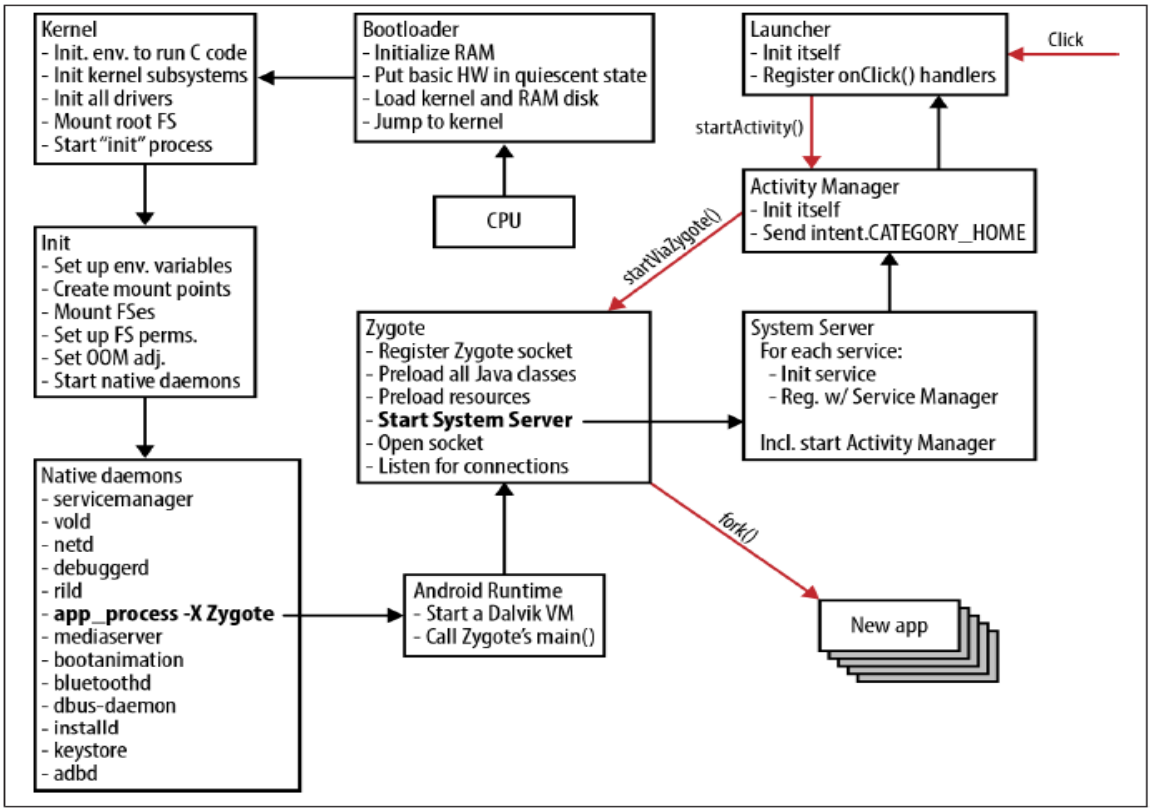

Figure 4. Android boot sequence [33].

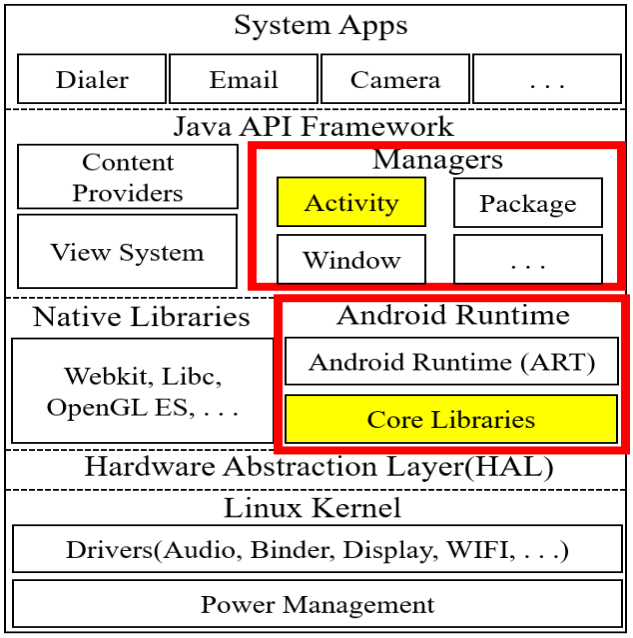

Figure 5. Android architecture.

4.2. User Authentication Using Peripheral Devices Architecture

UAPD is an automated process to protect against the SIA presented in the previous section. It identifies the user's device before the application is launched.

Figure 6 shows the overall process of UAPD, which is a detailed description of Figure 1. UAPD is a solution that reinforces the verification part of application execution. To verify a user's identity, each application is validated before its execution. This process relies on the Zygote, and the authentication routine is applied to all applications except system applications. For validation, the UAPD collects the necessary information once through the UAPD Initial Setting step in Figure 6.

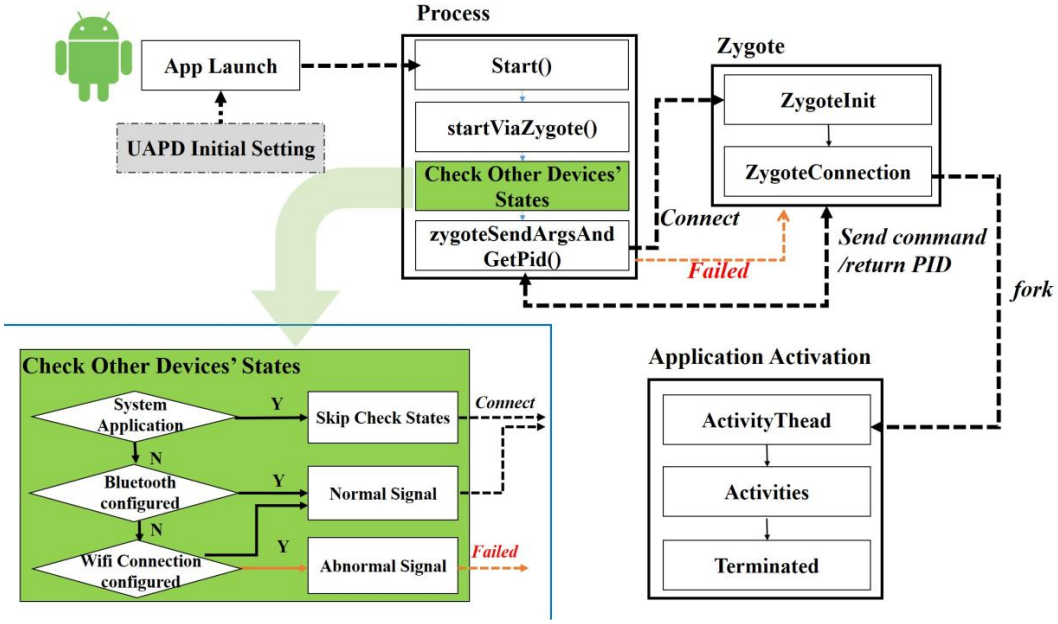

Figure 6. UAPD process.

Figure 7 shows the UAPD Initial Setting process required to prepare the authentication process. In this phase, the smartphone collects the user peripheral information. When a user runs the backup process on a new smartphone, the new smartphone requests the information for smartphone authentication from the devices that were used by the user. The new smartphone proceeds to authenticate the user using the received information. If there is no device for authentication around the user, the user's application will not run.

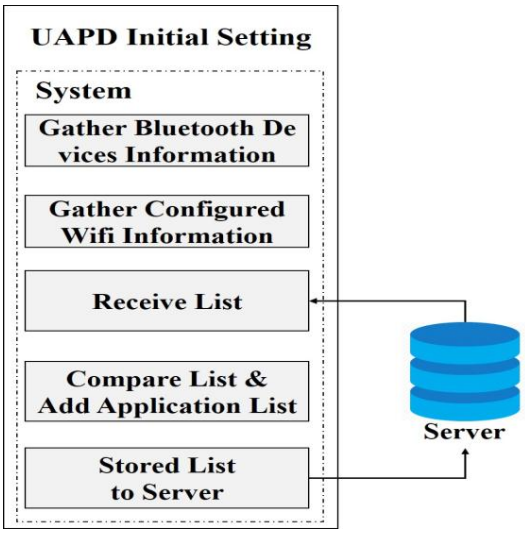

Figure 7. UAPD Initial Setting process.

Figure 8 shows the existing backup recovery method and backup recovery method using UAPD. The backup process involves creating the backup file, backup file movement, and application execution. Sometimes, some applications check the IMEI and IMSI values and initialize the data if the values differ from the values of the smartphone used by the user. However, as mentioned in the previous section, the attacker can change the information on the smartphone, such as IMEI and IMSI, and the original backup process cannot be protected against SIA. In the UAPD, as the UAPD Initial Setting and Check Other Devices States elements have been added to the original backup process, the attacker will not be able to perform SIA because there will be no peripheral devices for Check Other Devices States authentication, near the attacker.

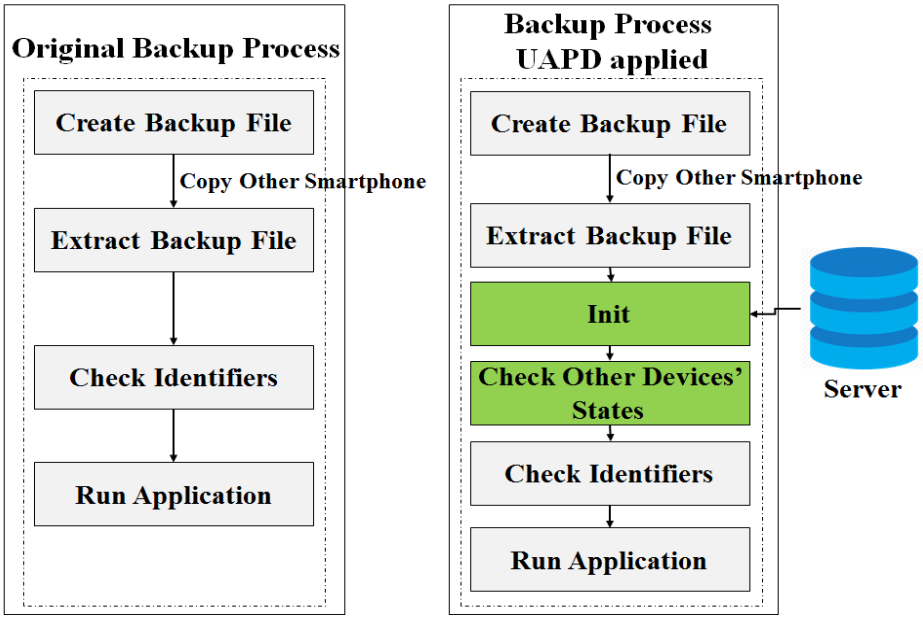

Figure 8. Original backup vs UAPD backup.

5. Evaluation

5.1. UAPD Overhead

There are two ways to defend Android applications against SIA: SIAD and the UAPD proposed in this paper. However, since SIAD goes through the encryption/decryption process, the execution time overhead is large. SIAD's performance depends on the database size of the application. Thus, in order to compare the performances of SIAD and UAPD, we used a test application provided by Android Studio on a Nexus 5X smartphone whose hardware specification is Snapdragon 808 CPU and 2 GB RAM.

Experiments were conducted to measure the execution time overhead, resource usage, and current consumption. Since the entire SIAD process encrypted and decrypted the databases existing in the application directory database folder, the execution time differed according to the size of the database of the Android application. Therefore, for a more precise measurement, the execution time was measured in the following manner, after increasing the size of the database by 1 MB.

1. Place the database file in the application directory.
2. Run the application to generate key values.
3. Quit the application and encrypt the database files in the application directory.
4. Run the application and decrypt the database files in the application directory.
5. Delete the key values and application data of the application for a new test.

The blue line superimposed on the horizontal axis in Figure 9 indicates the additional time required when running the application with UAPD. It requires an average of 0.005 s. When the application database is small, UAPD is approximately 30 times faster than SIAD. While the time consumed by SIAD increases with the increase in the size of the application's database, the UAPD consumes constant time. Eventually, it is shown that the UAPD is time-efficient over all time intervals, compared to SIAD.

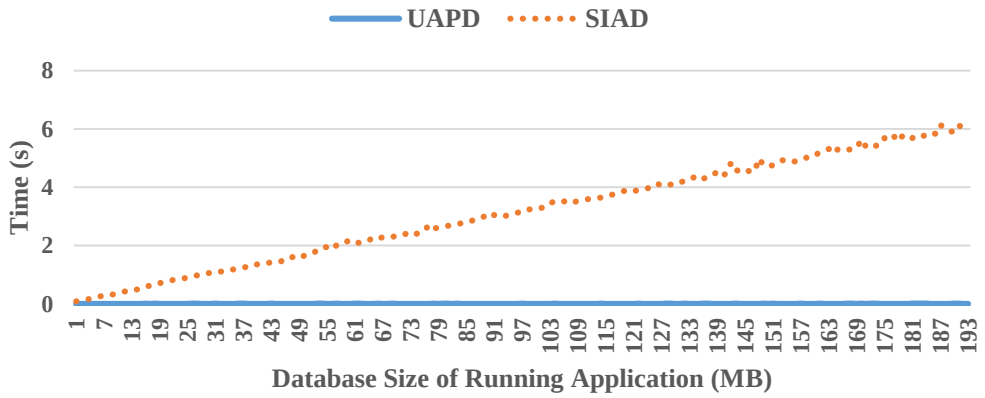

Figure 9. Time required for an application to run with UAPD.

Figure 10 shows the time-consumption ratio of UAPD and SIAD. As the size of the application's database increases, the efficiency can be seen to vary dramatically. For example, UAPD is 30 times faster than SIAD for a database size of 1 MB. Moreover, UAPD is 600 times faster than SIAD for a database size of 100 MB. The time-consumption ratio increases as the database size increases.

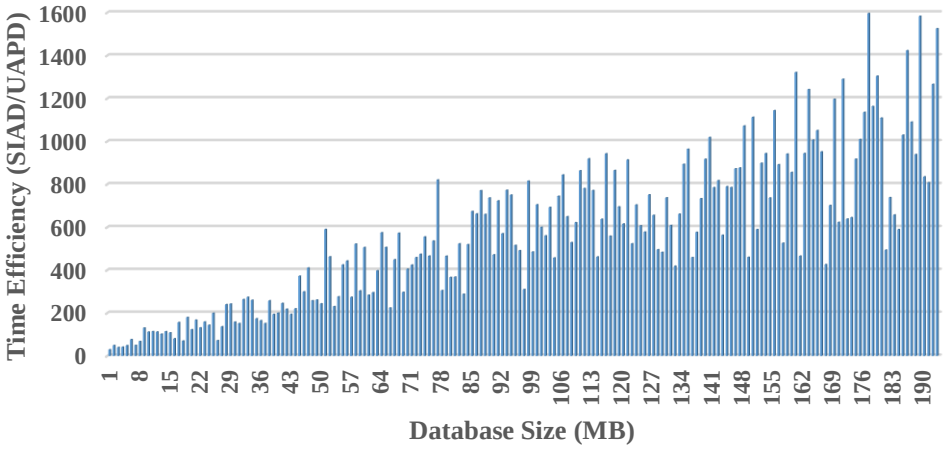

Figure 10. Time-efficiency comparison between SIAD and UAPD.

Figure 11 shows the test results illustrating the effects of the number of installed applications, especially, that of application packages, and other running applications on Android platform. By default, the number of installed packages is 79. We installed 41 additional applications that were in the top ranks of the application market, such as YouTube, Gmail, Map, Dropbox, and so on. Next, we tested the applications that had database sizes of 100 MB. As the numbers of installed and executed applications increased, there were no significant differences in the time overhead. Therefore, even if the number of packages increases, the efficiency of UAPD remains better than that of SIAD.

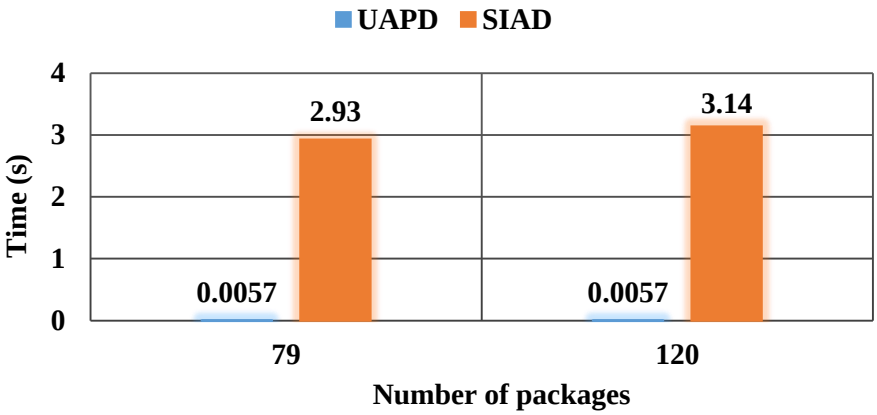

Figure 11. Time according to the number of application packages.

In Figure 12, we compared the CPU consumption of the original Android version (AOSP) from Android Open Source Project (AOSP) with the Android version UAPD implemented (UAPD) with Amazon applications using the top command. We repeated this process 100 times and averaged the results. UAPD (the Android UAPD implemented) and AOSP (the original Android) did not show significant differences in terms of CPU usage and memory consumption. In Figure 12, the CPU consumptions from zero to two seconds are similar, but there is a slight difference in the CPU consumptions from two to four seconds. Because Android applications are sometimes affected by the Android OS or other applications, the CPU consumption may increase in some time. For example, in a very rare case, the CPU consumption is 7% at one second, but 35% at two seconds. However, these values are not enough to affect the overall performance because the holding times of the values are very short. Therefore, applying UAPD incurs little CPU overhead.

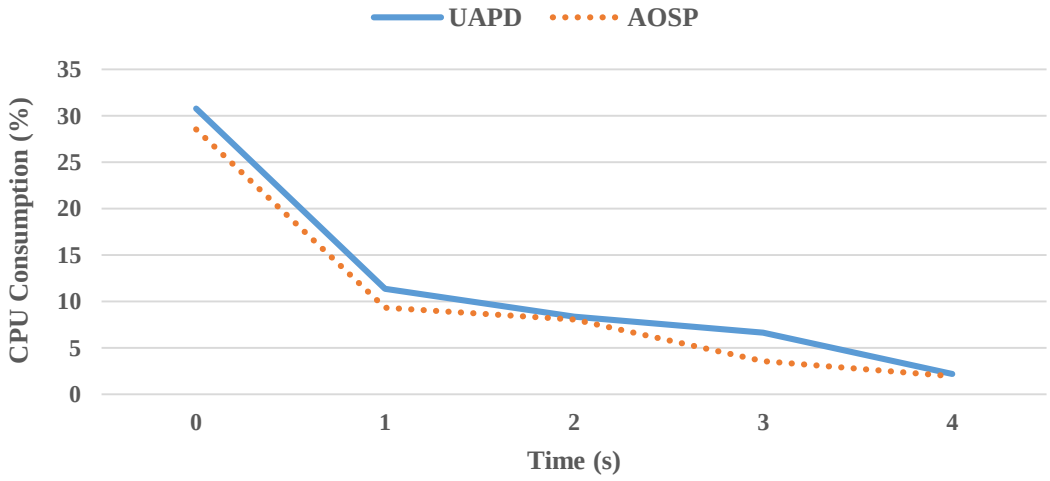

Figure 12. UAPD vs. AOSP (Android Open Source Project) for CPU usage.

Figures 13 and 14 show the measured values of the Virtual Set Size (VSS) and Resident Set Size (RSS) for the Amazon application. The values of VSS and RSS in UAPD and AOSP are similar; UAPD has little overhead.

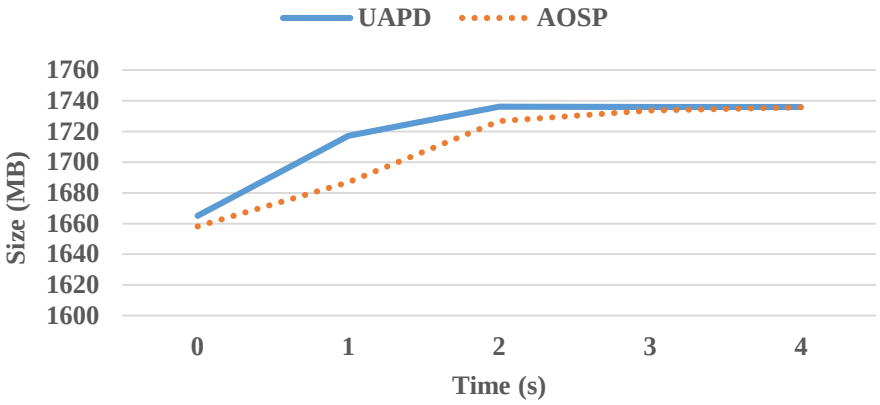

Figure 13. UAPD vs. AOSP for VSS (Virtual Set Size) usage.

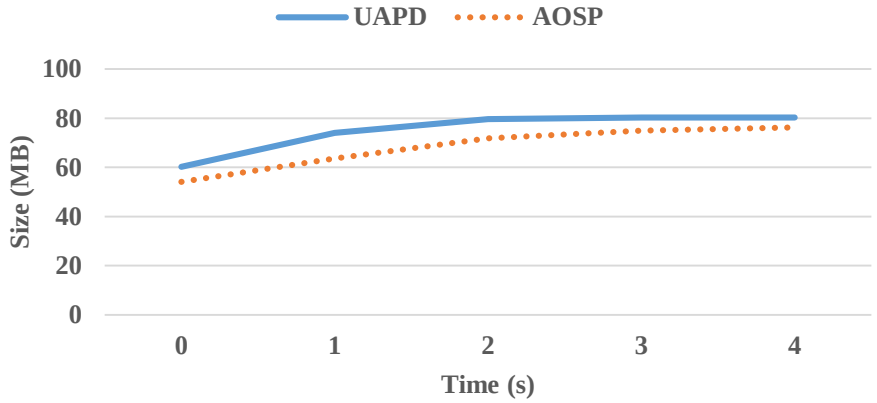

Figure 14. UAPD vs. AOSP for RSS (Resident Set Size) usage.

Figure 15 shows the memory usages for several applications in UAPD and AOSP. All three applications such as Amazon, Kakaotalk, and Facebook, running in the UAPD environment, have total memory sizes similar to those running in the AOSP environment; the application memory is often changed by external factors or application behaviors.

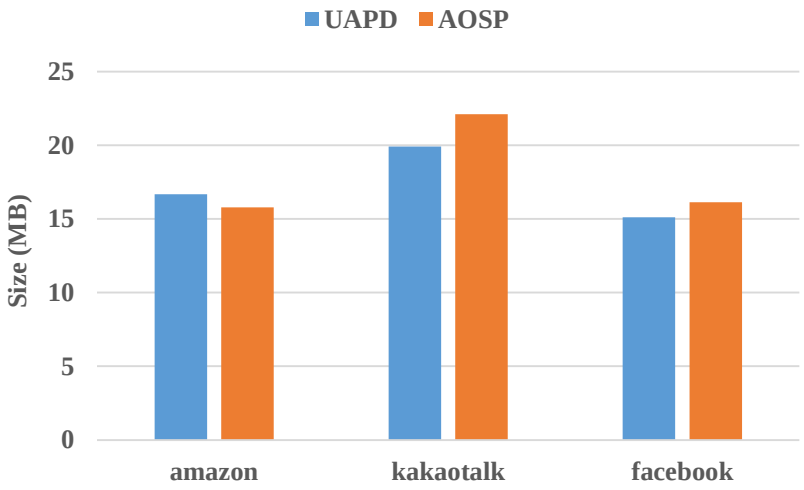

Figure 15. UAPD vs. AOSP in terms of memory usage.

5.2. UAPD Compatibility

Both the CipherSQL and SIAD mentioned earlier in the Related Works section have some compatibility with Android. CipherSQL is compatible with the versions 4 and 8 of Android. However, this is true only for the applications developed using CipherSQL. That is, as not all applications use CipherSQL, CipherSQL is not applied to all applications. SIAD modifies the Android Framework part, and therefore, can be applied to all applications.

On the other hand, there are different ways to terminate applications on Android, which rely on the developers creating the Android applications. A typical smartphone uses a custom OS of the original Android OS, which is provided by its manufacturer. Therefore, it is not possible to ensure that a routine added at the end of an application is applicable to all Android smartphones. The UAPD proposed in this paper modifies only the behavior of the application when it runs. Therefore, it can be applied to any Android OS. Moreover, the UAPD does not control all applications. Sensitive data such as card information, chat history, etc., of the applications we use are stored in the actual running application data area. As a result, it is possible to minimize the impact on traditional usage behavior by applying UAPD to commonly used applications other than system rights applications. The execution flow in the UAPD proceeds as follows: Start- > startViaZygote- > Check Other Devices Status- > zygoteSendArgsAndGetResult. Start calls startViaZygote. If an error occurs during execution, it is treated as a runtime exception.

5.3. UAPD Security

UAPD's goal is to defend against SIA. The previously proposed CipherSQL encrypted the database of the Android application so that, even if the attacker obtained the backup data, it was difficult to acquire the information of the application. However, the attacker could use the backup data to acquire and apply information such as the IMEI and IMSI from other applications that did not apply CipherSQL. In this case, even if the attacker did not know the key to decrypt the application data with CipherSQL applied, he/she could execute the application using the SIA method. On the other hand, UAPD defends against SIA through application execution via user authentication without additional encryption/decryption processes. With this approach, an attacker can acquire some application information from the backup data. However, the UAPD prevents the attacker from impersonating a user, by preventing the application from running. Applying this proposal has the disadvantage that an attacker can obtain some application information from the backup data. However, it does not allow the attacker to run applications. It also prevents the attacker from pretending to be a user.

There are two major security issues to consider when applying UAPD.

The first is when the user is authenticated via UAPD by verifying the peripheral connection. The information required for authentication must not be visible to the outside. The UAPD checks the configured WiFi or paired Bluetooth device on the smartphone. This will not require any input from outside the smartphone during UAPD. It also does not require a data transfer from the smartphone to the peripheral device. Therefore, UAPD does not involve data exchange for authentication, and is thus resistant to attacks on networks such as Man In the Middle attacks (MIMT) or packet modulation.

The second is related to an attacker circumventing the authentication process of UAPD using duplicated or modified information of the configured WiFi devices or paired Bluetooth devices. Information about WiFi configured on Android will be saved to the /data/misc/wifi/wpa_supplicant.conf file on the user's smartphone. The stored information includes the Service Set Identifier (SSID), Pre-Shared Key (PSK), Priority, and connection methods. To get these values, the attacker accesses the wpa_supplicant.conf file directly from the rooted device, or uses the application from an unrooted device. However, we do not consider rooted devices because rooted devices do not guarantee security and is an unusual situation. Moreover, the WiFi information will not be displayed using an application on an unrooted device, as the Android KitKat version displays the password of the WiFi as an encrypted hash value instead of in a plain text format. As a result,

the attacker cannot set the key value of WiFi and configure a WiFi environment similar to that of the user. On the other hand, regular users can synchronize and recover WiFi information that is configured on the Google Server by Google. Therefore, ordinary users can check the authentication of UAPD even if they backup, but an attacker cannot pass the authentication of UAPD using only the backup file. When the smartphone is paired with another Bluetooth device, a gatt profile is created in/data/misc/Bluetooth/(gatt_cache_number). However, as in the case of WiFi, only a rooted device can obtain information such as the key of a paired Bluetooth device, directly. Additionally, the information on paired Bluetooth devices are not backed up. Therefore, an attacker cannot arbitrarily create a Bluetooth device for authentication using only the backup file.

In Check Other Devices States, UAPD checks the connections to the devices configured or paired with the smartphone. The function that performs the connection checks uses the function that is used by the Android framework. Therefore, it has the same security as the connections made with WiFi and Bluetooth in the existing Android framework. Moreover, because they only check connections and do not perform any data exchange, they do not receive any data from external devices during the application execution. This ensures that you can prevent attacks that insert random codes.

6. Conclusions

In this paper, we showed that a copy of an application could be created by an attacker via SIA, which could subsequently be used to compromise security and obtain personal information. In particular, copies of WhatsApp, KakaoTalk, Facebook, Amazon, and Syrup were created with the same privileges as the user, and it was confirmed that they could cause secondary and tertiary damage.

We proposed UAPD to protect Android applications. As an improved defense of Android applications, which does not depend on the smartphone manufacturer, UAPD prevents data-replication attacks using IMEI, IMSI, or phone number, effectively. It was applied to and tested on a Nexus 5X smartphone to check its practicality. From the results, it was observed that the proposed UAPD was practical and useful, in terms of its performance, overhead, and compatibility.

UAPD is a proposal for companies that develop Android OSs, such as smartphone companies or Google. In the future, we plan to study the server communication protocol for UAPD use. Such a study will provide new user authentication methods.

Acknowledgments: This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Korea (2017R1D1A1B03034950) and by the BK21 Plus project funded by the Ministry of Education, Korea (21A20131600011).

Author Contributions: All the authors contributed equally to this work. All authors read and approved the final manuscript.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. All Products Require an Annual Contract Prices Do Not Include Sales. Google Play: Number of Downloads 2010–2016. Available online: https://www.statista.com/statistics/281106/number-of-android-app-downloads-from-google-play/ (accessed on 13 April 2018).
2. Armando, A.; Merlo, A.; Migliardi, M.; Verderame, L. Would You Mind Forking This Process? A Denial of Service Attack on Android (and Some Countermeasures). In *IFIP International Information Security Conference*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 13–24.
3. Lee, H.-T.; Kim, D.; Park, M.; Cho, S. Protecting Data on Android Platform against Privilege Escalation Attack. *Int. J. Comput. Math.* **2016**, *93*, 401–414. [CrossRef]
4. Sabt, M.; Traoré, J. Breaking into the Keystore: A Practical Forgery Attack against Android Keystore. In *European Symposium on Research in Computer Security*; Springer: Heraklion, Greece, 2016; pp. 531–548.
5. Davi, L.; Dmitrienko, A.; Sadeghi, A.-R.; Winandy, M. Privilege Escalation Attacks on Android. In *International Conference on Information Security*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 346–360.

6. Jung, J.-H.; Kim, J.Y.; Lee, H.-C.; Yi, J.H. Repackaging Attack on Android Banking Applications and Its Countermeasures. *Wirel. Pers. Commun.* **2013**, *73*, 1421–1437. [CrossRef]
7. Tam, K.; Feizollah, A.; Anuar, N.B.; Salleh, R.; Cavallaro, L. The Evolution of Android Malware and Android Analysis Techniques. *ACM Comput. Surv. (CSUR)* **2017**, *49*, 76. [CrossRef]
8. Wei, X.; Gomez, L.; Neamtiu, I.; Faloutsos, M. Malicious Android Applications in the Enterprise: What Do They Do and How Do We Fix It? In Proceedings of the 2012 IEEE 28th International Conference on Data Engineering Workshops (ICDEW), Arlington, VA, USA, 1–5 April 2012; pp. 251–254.
9. Joshi, J.; Parekh, C. Android Smartphone Vulnerabilities: A Survey. In Proceedings of the IEEE International Conference on Advances in Computing, Communication, & Automation (ICACCA), Dehradun, India, 8–9 April 2016; pp. 1–5.
10. Anglano, C. Forensic Analysis of WhatsApp Messenger on Android Smartphones. *Digit. Investig.* **2014**, *11*, 201–213. [CrossRef]
11. Al Mutawa, N.; Baggili, I.; Marrington, A. Forensic Analysis of Social Networking Applications on Mobile Devices. *Digit. Investig.* **2012**, *9*, S24–S33. [CrossRef]
12. Hacking Android Apps Using Backup Techniques. Available online: http://resources.infosecinstitute.com/android-hacking-security-part-15-hacking-android-apps-using-backup-techniques/ (accessed on 13 April 2018).
13. GitHub-Irsl/ADB-Backup-APK-Injection: Android ADB Backup APK Injection POC. Available online: https://github.com/irsl/ADB-Backup-APK-Injection/ (accessed on 13 April 2018).
14. Mutti, S.; Bacis, E.; Paraboschi, S. Sesqlite: Security Enhanced Sqlite: Mandatory Access Control for Android Databases. In Proceedings of the 31st Annual Computer Security Applications Conference, Los Angeles, CA, USA, 7–11 December 2015; pp. 411–420.
15. Paraboschi, S.; Bacis, E.; Mutti, S. Extending Mandatory Access Control Policies in Android. In *International Conference on Information Systems Security*; Springer: Cham, Switzerland, 2015; pp. 21–35.
16. Rachmawan, A. Developing Secure Android Application with Encrypted Database File Using Sqlcipher. Ph.D. Thesis, Universiti Teknikal Malaysia Melaka, Durian Tunggal, Malaysia, 2014.
17. Dai, Z.; Chua, T.-W.; Balakrishnan, D.K.; Thing, V.L. Chat-App Decryption Key Extraction through Information Flow Analysis. In *Systems Approach to Cyber Security*; IOP Press: Singapore, 2017.
18. Google Android: CVE Security Vulnerabilities, Versions and Detailed Reports. Available online: https://www.cvedetails.com/product/19997/GoogleAndroid.html?vendor_id=1224 (accessed on 13 April 2018).
19. Kim, J.; Jung, I.Y. Android App Protection Using Same Identifier Attack Defensor. In Proceedings of the IEEE International Conference on Intelligence and Security Informatics (ISI), Beijing, China, 22–24 July 2017; p. 201.
20. Kim, J.; Jung, I.Y. Private Data Protection of Android Application. In *Advances in Computer Science and Ubiquitous Computing*; Springer: Singapore, 2017; pp. 1470–1475.
21. Church, K.; De Oliveira, R. What's up with Whatsapp?: Comparing Mobile Instant Messaging Behaviors with Traditional SMS. In Proceedings of the 15th International Conference on Human-Computer Interaction with Mobile Devices and Services, Munich, Germany, 27–30 August 2013; pp. 352–361.
22. Ha, Y.W.; Kim, J.; Libaque-Saenz, C.F.; Chang, Y.; Park, M.-C. Use and Gratifications of Mobile SNSs: Facebook and KakaoTalk in Korea. *Telemat. Inform.* **2015**, *32*, 425–438. [CrossRef]
23. KaKaoTalk. Available online: http://www.kakaocorp.com/service/KakaoTalk (accessed on 13 April 2018).
24. Syrup. Available online: http://www.smartwallet.co.kr/index.do#about_syrup (accessed on 13 April 2018).
25. Jain, V.; Gaur, M.S.; Laxmi, V.; Mosbah, M. Detection of Sqlite Database Vulnerabilities in Android Apps. In *International Conference on Information Systems Security*; Springer: Cham, Switzerland, 2016; pp. 521–531.
26. Jiang, Y.Z.X.; Xuxian, Z. Detecting Passive Content Leaks and Pollution in Android Applications. In Proceedings of the 20th Network and Distributed System Security Symposium (NDSS), San Diego, CA, USA, 23 April 2013.
27. Hassanshahi, B.; Yap, R.H. Android Database Attacks Revisited. In Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, Abu Dhabi, United Arab Emirates, 2–6 April 2017; pp. 625–639.
28. Felt, A.P.; Chin, E.; Hanna, S.; Song, D.; Wagner, D. Android Permissions Demystified. In Proceedings of the 18th ACM conference on Computer and communications security, Chicago, IL, USA, 17–21 October 2011; pp. 627–638.
29. Tamma, R.; Tindall, D. *Learning Android Forensics*; Packt Publishing Ltd.: Birmingham, UK, 2015.

30. App Annie App Store Stats | Google Play Top App Matrix Overall. Available online: https://www.appannie.com/apps/google-play/matrix/?category=1&date=2017-06-07 (accessed on 13 April 2018).
31. Free Android Emulator on PC and Mac- Download Nox App Player. Available online: https://en.bignox.com/ (accessed on 13 April 2018).
32. Yaghmour, K. *Embedded Android: Porting, Extending, and Customizing*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2013.
33. Android Booting Sequence Explained. Available online: http://androidsrc.net/android-booting-sequence-explained/ (accessed on 13 April 2018).