Android-based Cryptocurrency Wallets: Attacks and Countermeasures

Cong Li
School of Software Engineering
East China Normal University
Shanghai, China
51184501123@stu.ecnu.edu.cn

Daojing He
School of Software Engineering
East China Normal University
Shanghai, China
djhe@sei.ecnu.edu.cn

Shihao Li
School of Software Engineering
East China Normal University
Shanghai, China
51174500101@stu.ecnu.edu.cn

Sencun Zhu
Department of Computer Science and Engineering
Pennsylvania State University
Pennsylvania, United States
sxz16@psu.edu

Sammy Chan
Department of Electrical Engineering
City University of Hong Kong
Hong Kong SAR, China
eeschan@cityu.edu.hk

Yao Cheng
Institute for Infocomm Research
*A*STAR*, Singapore
chengyao@i2r.a-star.edu.sg

***Abstract*—The security of cryptocurrency wallets is directly related to the security of personal assets. However, due to the design defects of mobile operating system and cryptocurrency wallets, security incidents of cryptocurrency wallets occur frequently, causing irreversible losses to users' assets or privacy. In this paper, we study the security risks of Android-based cryptocurrency wallets. We establish the adversary model, analyze the attack surface originated from the Android OS, and demonstrate several attack vectors by conducting experiments on multiple popular cryptocurrency wallets in Google Play Store. Finally, we present several security defense strategies in response to the security risks.**

***Index Terms*—Cryptocurrency wallets, Android application security, Permission, Blockchain.**

I. Introduction

Blockchain is a distributed and append-only ledger originated in 2009 [1]. Since the introduction of the first blockchain-based application - Bitcoin, cryptocurrencies have experienced rapid growth. The total market value of cryptocurrencies has reached over $250 billion in early June 2020.

Cryptocurrency systems (e.g., Bitcoin and Ethereum) use the private key as the only credential for account management, so owning the private key means the full control over the account. Once a cryptocurrency transaction is signed by the private key, it will be considered legal and irreversible. There are two important causes for the loss of digital currency assets, which are the theft and the loss of private keys. Therefore, the security of cryptocurrency assets, in other words, is actually the security of the private key.

To address the private key loss issue, recently digital wallet applications have been deployed to store private keys, and they are also responsible for producing and managing private keys, as well as for managing the transactions of digital currencies. Due to the critical role of private key, highly secure private key generation algorithms have been implemented. For example, the key derivation algorithm PBKDF2 iterates many times to generate more secure private keys, which prevent not only the "brute force cracking", but also the "rainbow table attacks".

A major deployment platform for cryptocurrency wallets is mobile phones because of the convenience. Among the mobile phone platforms, currently Android leads with around 75% global market share. In [2], several Android-based cryptocurrency wallet apps have been studied based on static code analysis and network traffic analysis, which revealed a few common mobile app implementation vulnerabilities, including suspicious permission requests. On the other hand, malicious apps on smartphones might also exploit system vulnerabilities to steal sensitive information [3]. The vulnerabilities in the cryptocurrency wallets and the lack of users' security awareness might also lead to security credentials being stolen and the parameters of transactions being tampered with.

In this work, we take a systemic approach to study the security vulnerabilities of Android-based cryptocurrency wallet apps. We establish the adversary model, analyze the attack surface originated from the Android OS, and demonstrate several attack vectors by conducting experiments on multiple popular digital wallets in Google Play Store. Finally, we present several security defense strategies in response to the security risks.

Our contributions are summarized below:

- We identify security vulnerabilities in several popular cryptocurrency wallets in Google Play Store, which could be exploited by malicious attackers.
- We design several proof-of-concept attacks based on the identified vulnerabilities, and our attack experiments demonstrate the validity of our analysis.
- We suggest corresponding security measures to reduce the attack surface.

978-0-7381-0495-9/20/$31.00 ©2020 IEEE
DOI 10.1109/Blockchain50366.2020.00010

The rest of the paper is organized as follows: Section II explains the concepts of cryptocurrency wallets, Android permission mechanisms and adversary model. Section III introduces the attack surface. Section IV presents the attack experiments and their results. Section V discusses possible defense mechanisms. Section VI discusses related work, followed by Section VII that summarizes our research and provides direction of future work.

II. Preliminary Knowledge and Adversary Model

In this section, we briefly introduce the necessary background to better understand this paper.

A. Cryptocurrency Wallets

In our study, we consider cryptocurrencies that are implemented with decentralized control, typically based on the blockchain technology. Cryptocurrency wallets are the applications specifically used to manage the digital assets, including creating account address, managing cryptocurrency transactions, supporting queries of transaction records and other basic financial services.

Wallet applications create one or more wallet addresses by hash function, and each address has a key pair: private key and public key. The public key is primarily used for external transactions. Each transaction must be signed with the private key to prove the ownership of the assets. Most wallets back up their private keys and store them in encrypted form. In Ethereum, private keys can be represented in hexadecimal plaintext, mnemonic words or keystore+password. Specifically, based on the mnemonic and salt (the string "mnemonic"), one can call a specific hash function to restore the private key. Keystore is the encrypted form of the private key for storage, so the private key can be obtained later by the keystore and password. The mnemonic generates the keystore indirectly by generating the private key.

B. Android Permission Mechanism

Android system provides many APIs for applications to access mobile phone hardware, user data and communication components. While Android APIs provide rich network and sensor data to applications for many useful functionalities and services, some of them may be invoked to access sensitive user information and cause system damage if not used properly. Android employs a permission mechanism to protect users' security and privacy. The permissions that can be requested by a third-party app can be divided into two categories: normal permissions, which generally do not involve user privacy and therefore require no user authorization (e.g., read accelerometer sensor); dangerous permissions, which involve user privacy and require user authorization (e.g. access to sdcard, contacts etc.). To control the use of system resources by apps, each app needs to declare the required permissions in its *AndroidManifest.xml* file. Roughly speaking, before Android 6.0, the system automatically asks the user to grant all dangerous permissions for the app at install-time; after version 6.0, dangerous permissions are requested at run-time.

C. Adversary Model

The basic goals of adversaries are to break the confidentiality, integrity, or availability properties of the data in the wallets. Specifically, they aim to steal users' accounts and passwords, and corrupt transactions to steal users' assets.

In particular, we assume that:

1) The adversary has no special knowledge on the target: that is, the attacker does not know the user account password before carrying out the attack.
2) The adversary has the ability of luring the user to install some malicious applications, and such malicious applications can acquire necessary permissions.

There are two possible approaches for adversaries to install a malicious app on an android device. In the first approach, the adversary develops a legitimate app (e.g., a useful utility app) and uploads to the official app market–Google Play Store. When such an app is installed, the adversary can request and obtain the BIND_ACCESSIBILITY_SERVICE permission through performing "clickjacking attack", and then install the malicious app on the device through sideloading or drive-by download [4]. In the second approach, the malicious app is distributed through an unofficial link. For example, the adversary may send some seductive links to lure users to install the malicious app.

III. Attack Surface of Cryptocurrency Wallet

A. Reading or Writing External Storage

The Android operating system divides storage into internal and external storage. In the absence of root permissions, by default, files saved in internal storage can only be accessed by the app itself, so it is suitable for saving the app's private files in the internal storage. Apps with the right to read external storage READ_EXTERNAL_STORAGE or write to external storage WRITE_EXTERNAL_STORAGE can read or write any data in the external storage without raising the user's attention. External storage permissions are widely used by various applications. Some private information, including photos, videos, documents, screenshots and others, are usually stored in external storage.

In order to better control the use of external storage, Android 6.0 and above require applications to apply for permissions at runtime. Users will be prompted when the app is running, not just when it is installed, asking if the app is allowed to read and write to external storage. However, the app may continuously ask the user to give permission when the user refuses to authorize; otherwise the user is denied to use the application. In this way, the user is forced to grant the permission. Because this permission is requested by most applications, users are likely to allow applications to read and write to external storage. In addition, if the application sets its *targetSdkVersion* to 22 or below, which means that the

target Android version of the application is 5.1 or below, the system will consider that the application does not conform to the Android 6.0 OS, so it still uses the old-style permission management in which there is no permission request popup at run-time.

B. Backups Files

In order to prevent accidental loss of application data caused by device damage, user mis-operation and other unexpected circumstances damaging the availability of data, the Android OS comes with the application backup service, which can backup the application and its data to a local file or a cloud server for subsequent recovery. The system backup file may contain data from the app's private directory. Some systems back up data without encryption to an external storage, allowing apps with the permission of reading external storage to acquire private data from other apps. The data often contain more sensitive information. If the app itself does not encrypt the data, its backup file will break the confidentiality of sensitive information.

C. Showing Floating Window

Applications with SYSTEM_ALERT_WINDOW permissions can draw anything on the screen. Benign apps can use this feature to perform legitimate actions such as creating a floating window to provide certain convenience to users or change the brightness of the screen display to meet user requirements. A malicious application may draw an overlaid window on other applications to tamper with the user interface, such as asking users for passwords, modifying transactions records or masking what is actually performed.

D. Reading or Writing Clipboard

Running Android apps can read and write the clipboard through the ClipboardManager component provided by the system, even without being granted any special permissions or notifying the user. Benign apps can utilize this feature to quickly copy self-defined data to the clipboard, or analyze data within the clipboard to simplify operations and improve the user experience. However, a malicious app can exploit this feature to monitor the contents of a user's clipboard and steal or replace sensitive data.

E. USB Debugging Service

USB debugging service is for developers to debug and test apps. It usually requires the device to be connected to a computer via a USB cable. Since the computer and the Android device are physically connected, launching attacks against the connection is not easy. However, Android supports wireless debugging from a computer within the same local network. Wireless debugging needs to be turned on in advance, either through an application with root permissions on the device, or by using the command "*adb tcpip*" during the last wired USB debugging.

Wireless debugging makes debugging possible without users' awareness. The *adb* shell command can run executables in the Linux shell environment of Android devices with shell privileges. One can make the command running in the background by tailing "&" to the command. Linux also provides a *nohup* command that keeps the program running after the user disconnects. With these features, the attacked device can run a process with certain privileges in the background, perform privileged operations used in USB debugging. The process can also communicate with other apps on the device, for example, through sockets, and run instructions sent by other apps, thus promoting other apps to shell privileges.

IV. Attack Experiments

In this section, we present attack experiments exploiting the above mentioned vulnerabilities. We demonstrate the validity of our analysis by breaking wallet security mechanisms. We choose attack targets from Google Play Store. We search the keywords "Bitcoin wallet" and "Ether wallet" to download the top 5 wallet apps. The app "Blockchain Wallet" appeared in both searches, so we only keep one record. We also removed the cloud-based Ethereum Wallet (by Freewallet) because their private keys are stored in the cloud. This results in a total of eight apps as listed in Table I.

We carry out experiments from three different aspects: reading backup files, identifying user transaction information and changing transfer address, stealing user privacy information based on accessibility service on Huawei honor 7x, MI 8 and BlueStacks 3.1. The phone system is Android 8.0 and 9.0. The details of the attack experiments are discussed below.

A. Acquire Sensitive Information from Backup File via USB Debugging

When Android software developers perform testing on Android devices, they need to keep USB debugging mode on when they connect an Android device with a computer for backup or development operations. The attacker can use the computer to operate on the user's device if malware has been installed in the computer. Meanwhile, the user will not receive any notification. The parameter *android:allowBackup=true* in the *AndroidManifest.xml* profile, which is included in every app, is true by default. If it is set to false, we cannot back up the app's data by the adb command *backup*. Even if that does not work, an attacker could also attempt to get local backup files on the mobile device. Some local backup implementations may ignore the *android:allowBackup=false* configuration in the *Android Manifest.xml* profile. In the Android-based MIUI system, traversing each file in local backup directory "/sdcard/MIUI/Backup/AllBackup/backup/backup time/", the file names betray the corresponding application of the backup file.

After the backup file is obtained, it is unpacked according to the backup file format. Then, according to the correspond-

TABLE I: Target cryptocurrency wallets

No	wallet name	Package Name	Version	Google Play Installations
A	Blockchain Wallet	piuk.blockchain.android	6.29.1	5,000,000+
B	Bitcoin.com Wallet	com.bitcoin.mwallet	5.12.3	1,000,000+
C	Bitcoin Wallet	de.schildbach.wallet	7.21	1,000,000+
D	BitPay	com.bitpay.wallet	6.3.5	500,000+
E	Mycelium Bitcoin Wallet	com.mycelium.wallet	3.0.0.23	500,000+
F	MEWconnect	com.myetherwallet.mewconnect	1.0.6	100,000+
G	Coinbase Wallet	org.toshi	18.3.222	100,000+
H	Trust Wallet	com.wallet.crypto.trustapp	1.7.055	100,000+

ing formats of private key, mnemonic word, keystore and WIF (Wallet Import Format), each of them can be extracted. Sensitive information can be analyzed by regular expression matching.

As shown in Figure 1, after executing the command *adb backup* on the computer, for wallets B and D, we can access the backup file. Since wallets B and D do not set a PIN, the mnemonic word can be acquired from the backup file. The other wallets cannot be backed up because the parameter setting for *android:allowBackup = false*.

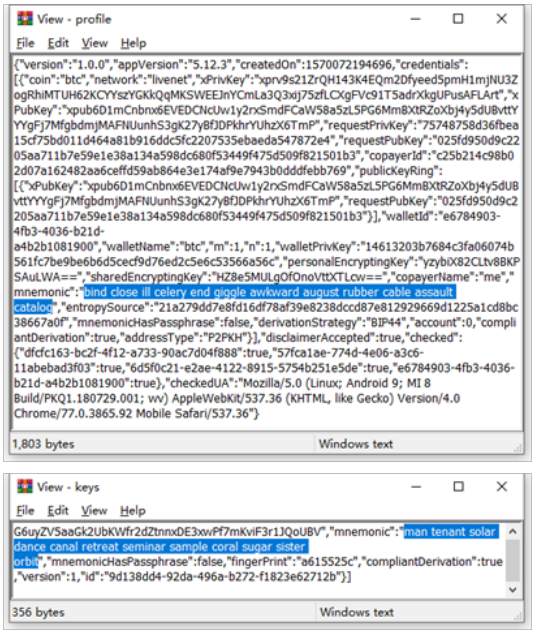

Fig. 1: Mnemonics stolen from backup files.

B. Identify Sensitive Information in Screenshots

Some users may save screenshots containing sensitive information locally, this gives the attacker an opportunity to extract the senstive information. As mentioned in Section II-C, the attacker can build a malware and acquiring READ_EXTERNAL_STORAGE permission. When the malware finds a screenshot related to a wallet, OCR (Optical Character Recognition) can help converting the image into text and extract information from it. This is described in more detail below:

1) Detecting the image changes: Android provides content provider ContentProvider and the corresponding ContentResolver mechanism to share information across apps. By using ContentResolver to register a listener with MediaStore.Images.Media.EXTERNAL_-CONTENT_URI, the malware will be notified when the images on the device changes.
2) When a new screenshot is generated, the listener process queries the latest picture of the media library: the *handleWindowContentChange()* function is triggered, and the ContentResolver is used to obtain the latest picture. Then, according to the size of the picture, the keywords of storage path (e.g., wallet package name), the wallet screenshots are identified.
3) Extracting sensitive information in the screenshot: First, the identified wallet screenshot is converted into byte form. It is then passed to Tencent OCR (or any other optical character recognition tools) to extract the sensitive information. The result is returned as a string in JSON format.

For most cryptocurrency wallets, generally, WindowManager.LayoutParams.FLAG_SECURE will be set in Android activity for security. It means that this app prohibits screenshots and screen recording. However, there are still wallets that allow screenshots, like wallets B, D and G. So, the attacker can capture private key, mnemonic etc in this way.

C. Recognize and Replace Sensitive Information

Generally, the private key of the cryptocurrency wallets, keystore and other sensitive information consist mostly of random characters and are too long to be memorized. So it is likely that the user will input them by copying and pasting. Of course, there are some frameworks providing automatic filling of private keys, such as AutoFill Framework [5], openYOLO [6], but they are exposed to the risk that will leak user's passwords [7].

Android provides *ClipboardManager* for applications to completely monitor, read and modify the contents of the clipboard. By adding a listener to the clipboard, the listener is notified when the contents of the clipboard change. The malware can exploit this feature to extract sensitive information from the clipboard according to the format of private key, mnemonic, keystore, WIF format private key.

For wallets B and G, sensitive information is automatically acquired when the user copies sensitive information such as mnemonics and private key. Figure 2 demonstrates that 12 complete mnemonics are successfully obtained by this attack.

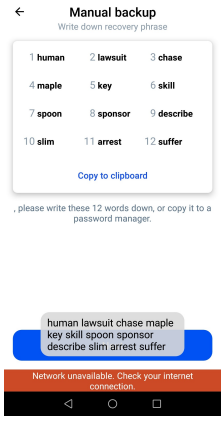

Fig. 2: Stealing sensitive information from clipboard. For clear demonstration, floating window is designed to display the obtained mnemonics.

Users often copy and paste information such as the wallet address during the transaction process. Implementing the same operation as mentioned above, the content in clipboard can be acquired easily. By matching the rules for the contents of the clipboard as shown in Table II, when the wallet address is discovered, it can be replaced by the attacker's address using the relevant interface provided by *ClipboardManager*. The attacker may upload the original address to his own server, and the server can select the most similar address among all the account addresses owned by the attacker (or the one sharing the same first and last characters as the original address) and send it back to replace the original address. In this way, it may reduce the possibility of being noticed by the user.

TABLE II: Account Address Extract Regular Expression

Type	Regular rules
Bitcoin address	(bc1—[13])[a-zA-HJ-NP-Z0-9]{25,39}
Ethereum address	0x[0-9a-fA-F]{40}

For all wallets, when the user copies the wallet address, it can be automatically replaced with the address set by the attacker without any warning. Screenshots of this attack are shown in Figure 3.

D. Steal Sensitive Information by Accessibility Service

Accessibility service provides permissions to get most of the information displayed on the screen, except the password textbox, which is protected by default. However, there are methods to obtain the password, such as monitoring the user's input through soft keyboard. These soft keyboards often come from different third-party input methods. In a popular password management software keepass2android, when its own virtual keyboard is clicked, the key slot will produce visual feedback, and trigger TYPE_WINDOW_-STATE_CHANGED events. When the password of a wallet is entered, the attacker can infer the user's password by listening to the click event.

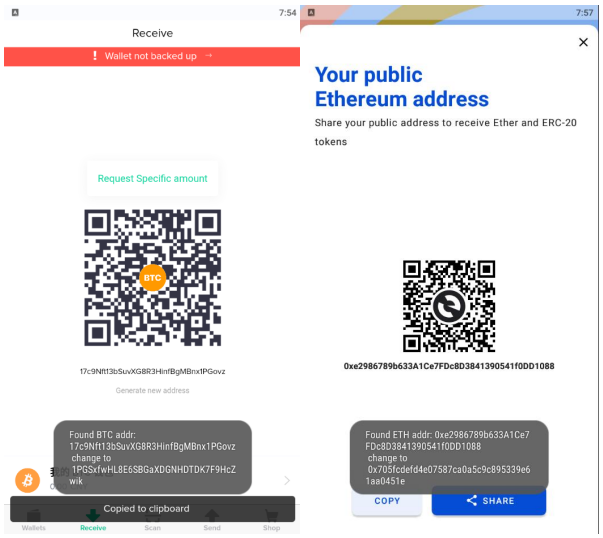

Fig. 3: Replacing transaction information in native clipboard. We display the malicious address and the original address using the floating window during the transaction address replacing attack. The screenshots show the effect of BTC and ETC currency trading address after being attacked.

Assuming keepass2android's input method is used when entering the password, an attacker firstly acquires the keyboard layout of keepass2android, which is unchanged, so the attacker can record and infer the clicked Key-value by location over time. Wallets A and F allow to enter passwords using third-party input methods, while the remaining ones must use their own secure keyboards. A screenshot of this attack is shown in Figure 4.

V. Possible Defense Mechanisms

Basing on the experiments and analysis in the previous section, as well as the fact that wallet applications do not have system privilege, we put forward in this section some protective measures to enhance the security protection of wallets.

For the insecure services provided by the Android OS, a detection method can be adopted to remind the user of the risk when they are turned on, and guide the user to turn it off. In view of the possible insecure operation, we will consider application scenarios and take a preventive approach to protect the user from carrying out the dangerous operation.

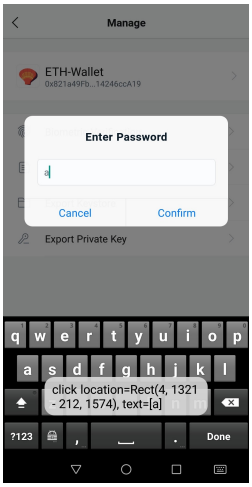

Fig. 4: Inferring input password from third-party keyboard using Accessibility service.

A. Running Environment Detection

When the wallet is running, by detecting whether certain hazard features are enabled, users should be alerted about the hazards and provided an option to disable them. In this way, users have a clearer understanding of the security risks, and can turn off the hazard features that they do not really need, reducing the possibility of security incidents from occurring. For most insecure features, we can utilize the Android System Settings feature to turn them on and off. The system settings are divided into system, secure and global, and we can use *Settings.System.getString()*, *Settings.Secure.getString()* and *Settings.Global.getString()* to acquire these states without requesting for special permissions. When the hazard services are activated, the user should be alerted about the dangers that may be caused and be urged to turn off them. Programmatically modifying system settings requires WRITE_SETTINGS or even WRITE_SECURE_SETTINGS permissions, which is not available to third-party apps in recent Android versions. The only practical way is directing users to manually modify system settings. Android provides the *startActivity()* function to start another activity in one activity with an argument *intent*. After setting the target of intent to a specific value (the target component name of the jump), we can enter the corresponding interface to complete some operations. There are some values of the actions, jumping to the system settings interface in the *android.provider.Settings* class. These features allow users to quickly jump to the appropriate settings interface, which make it easier for users to turn off features.

Note that if we use a window that can be immediately clicked to close by users to show users the prompting message, attackers can use USB debugging, Accessibility Services, and other dangerous features to simulate user operations, closing the prompting window directly. Therefore, a prompting window that stays on for a sufficiently long time may be a better choice. Although this approach is useful, it can have a negative impact on the user experience and needs to be weighed.

1) ***USB Debugging****:* The value of the *adb_enabled* item set by the global class, represents the state of USB debugging, the value of 1 representing "on" and 0 representing "off". When setting.ACTION_APPLICATION_DEVELOPMENT _SETTINGS is used as the intent of the action, we can guide the user to jump to the developer options page and then guide the user to turn off the USB debugging option. For more accurate reaction, it should take into consideration the user's purpose, whether to use USB debugging mode or not.

2) ***Accessibility Services****:* The value of the *accessibility_enabled* item set by the secure class represents the accessibility service state of the whole system with 1 representing switched on and 0 representing off. In addition to the whole system's accessibility switch, a user also needs to turn on each accessibility service individually. It is possible that the whole system accessibility switch is turned on, but no individual accessibility functions are activated. Thus, one can get the package and class names that have been enabled by acquiring the value of the *accessibility_services* item set by the secure class, and if it is empty, no accessibility features are actually started.

In order to prevent users from receiving warnings caused by the use of normal accessibility service, such as the system's built-in accessibility service talkback for the visually impaired, we can preset a white list of package names for legal accessibility services, verifying whether the accessibility services that have been activated belong to the white list, then judge the legitimacy. However, a malicious application may also modify its "packageName" to that of a benign app, attempting to bypass the white-list based detection. As Android OS applies application signatures to prevent this problem, we can use *getPackageManager().GetPackageInfo("packagename", PackageManager.GET_SIGNATURES).signatures* to get the digital signature information of the package name and compare it with the legitimate digital signature to determine whether it is a fake application.

Considering that accessibility service is still used by many normal applications for automated operations, the wallet can further judge whether accessibility service can read its interface, and ignore it if it is not actually accessible. The wallet can obtain the APK package name of the application running the accessibility service, and parse its content to obtain the configuration file of the accessibility service. It then identifies the *packageNames* item in the configuration to determine whether the accessibility service is listening to the wallet application, and checks the configured *android: canRetrieveWindowContent* to determine whether it can obtain the screen content. The complete detection logic for accessibility service is shown in Figure 5.

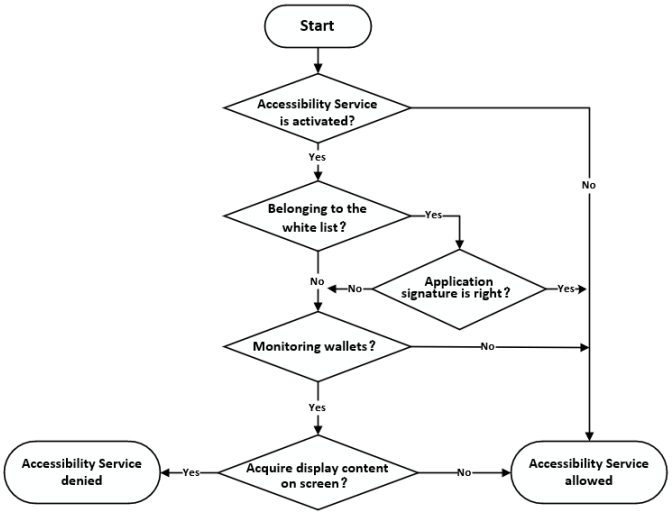

Fig. 5: Accessibility Detection Flow Chart.

When the wallet finds a suspicious accessibility feature, it should prompt the user to temporarily turn it off. At this time, it should start the activity with Settings.ACTION_ACCESSIBILITY_SETTINGS as the intended action, and jump to the accessibility management page.

B. Hazardous operation prevention and detection

Besides detecting the wallet running environment and urging the user to turn off the dangerous features, the wallet can also prevent or detect hazardous operations to reduce the possibility of being attacked. Saracino *et al.* [8] leveraged the behavior of smartphone users to detect the existence of malware. Other works [9] focused on detecting malicious apps by detecting the mismatch between runtime permission request and GUI-based context. When the user is performing a hazardous operation, the wallet should warn the user of the degree of risks of the action they are taking and provide remedial measures for user's reference. For common hazardous operations, there are several methods to prevent and detect.

Since the clipboard can be read by any applications, the wallet should warn users in advance of the possible harm of copying and pasting data. Specially, we should prohibit copy and paste in some extremely sensitive circumstance, forcing users not to use the clipboard to handle sensitive data. However, some of the sensitive information involved in cryptocurrency wallets, such as keys and keystores, are mostly very long and difficult to type. Prohibiting copy and paste will cause greater difficulties for users' input and backup. So, the wallet should use two-dimensional code, NFC, and other methods to facilitate user input.

Malicious apps can obtain private information such as keys in the clipboard through *ClipboardManager*, as cryptocurrency wallets do. When the wallet security mechanism finds private information, it should warn users that the clipboard can be read by other third-party applications, and remind users to pay attention to the safety of funds since those private information may have been leaked.

Similarly, wallets can also listen to account address replacement operations that occur in the clipboard. When an account replacement is detected, the wallet security mechanism should warn the user that the account address may be replaced by a malicious application. At the same time, it should display the wallet addresses before and after replacement, prompting the user to immediately terminate the transfer if it is not the user's own operation, and pay attention to the device security.

VI. Related Work

The security risks of Android cryptocurrency wallets come from two aspects: (1) the security vulnerabilities from the design and implementation of the cryptocurrency wallets; (2) the features and weaknesses of the Android OS. In the past years, relevant explorations have been made on the analysis of security weaknesses in the Android operating system. Diao *et al.* [10] used the interrupt processing mechanism to infer the user's unlock pattern and foreground application state without any permissions. Xu *et al.* [11] leveraged mobile phone accelerometer sensor and orientation sensor as the side channel to infer user's sensitive tapping input such as PIN and credit card numbers. Chen *et al.* [12] used the shared memory to infer the state change of the graphical interface of the application, without the need of any permissions. Ren *et al.* [13] proposed UI attacks to attract users to enter their credentials into fake/malicious UIs. Bentley *et al.* [14] discovered how to use Accessibility Service to install any desirable apps, thereby causing security threats of privacy information. Reardon *et al.* [15] described how to circumvent the permission model and gain access to protected data and system resources without user consent by using both covert and side channels. These works have shown that these attacks are practical and affect millions of Android applications.

Rather than focusing on the security of general Android applications, there are also a few research work focusing on Android-based cryptocurrency wallets. Bamert *et al.* [16] used a combination of hardware and software to create the Bitcoin hardware token BlueWallet. Connected to the wallet via Bluetooth, it can be used to securely sign Bitcoin transactions. Goldfeder *et al.* [17] proposed the first threshold signature scheme compatible with the ECDSA signature used in Bitcoin to enhance the security of bitcoin wallets. For the security of digital wallets, Androulaki *et al.* [18] proposed some ways to improve the security of Bitcoin wallets and user privacy information. Some of the ideas are also effective for mobile wallet security. These methods address the security challenges for mobile devices from different perspectives. In addition to improving the security of the cryptocurrency wallets, wallets can also change the method for backing up sensitive information. Okpara *et al* [19] proposed to use smart cards for wallet backup, reducing the threat of untrusted Android OS.

The work most relevant to ours are [2] and [20]. The first one [2] exploits static code analysis and network data analysis to evaluate cryptocurrency wallet apps to discover common security vulnerabilities in Android apps. However, it does not demonstrate any experiments on any applications and explain how they work in practice. The second one [20] presents some preliminary results on the security risk of cryptocurrency wallets and the vulnerability of Android OS, and then verifies its correctness through experiments. Our work proposes a more comprehensive attack surface, conducts relevant attack experiments, and further presents the corresponding defense strategies.

VII. Conclusion

In this paper, we have analyzed the security of cryptocurrency wallets. Studying from the perspectives of backup files, Android clipboard and Accessibility Service, we have identified several vulnerabilities originating from cryptocurrency wallets and Android's permission management. We have prototyped malicious apps which can easily break the security mechanisms of popular wallets in Google Play Store, steal the input of the secure keyboard and capture the user's private information. To address these vulnerabilities, we have proposed some practical security strategies to improve the security of cryptocurrency wallets.

Due to the differences in the security threats of various cryptocurrency wallets, we mainly studied in this paper the wallets running on the Android platform. In the future, we can analyze features of different platforms and extend our work to IOS, PC and even hardware wallets.

Acknowledgment

This research is supported by the National Key R&D Program of China (2017YFB0802805 and 2017YFB0801701), the National Natural Science Foundation of China (Grants: U1936120, U1636216), Joint Fund of Ministry of Education of China for Equipment Preresearch (No. 6141A020333), the Fundamental Research Funds for the Central Universities, and the Basic Research Program of State Grid Shanghai Municipal Electric Power Company (52094019007F). Daojing He is the corresponding author of this article.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Cryptography Mailing list at https://metzdowd.com*, March 2009.

[2] A. Lero, J. Buckley, and A. Gear, "Privacy and security analysis of cryptocurrency mobile applications," in *Proceedings of the Fifth Conference on Mobile and Secure Services (MobiSecServ), Florida, USA, March 2-3*, 2019, pp. 1–6.

[3] Y. Zhou, X. Zhang, X. Jiang, and V. W. Freeh, "Taming information-stealing smartphone applications (on android)," in *Proceedings of the 4th International Conference on Trust and Trustworthy Computing, Pittsburgh, PA, USA, June 22-24*, 2011, pp. 93–107.

[4] Y. Fratantonio, C. Qian, S. P. Chung, and W. Lee, "Cloak and dagger: From two permissions to complete control of the ui feedback loop," in *Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, May 22-26*, 2017, pp. 1041–1057.

[5] Google. Autofill framework. [Online]. Available: https://developer.android.google.cn/guide/topics/text/autofill

[6] Google'. Openyolo for android. [Online]. Available: https://openid.net/specs/openyolo-android-03.html

[7] S. Aonzo, A. Merlo, G. Tavella, and Y. Fratantonio, "Phishing attacks on modern android," in *Proceedings of the 25th ACM SIGSAC Conference on Computer and Communications Security (CCS '18), Toronto, Canada, October 15-19*, 2018, pp. 1788–1801.

[8] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "Madam: Effective and efficient behavior-based android malware detection and prevention," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 1, pp. 83–97, January 2018.

[9] H. Fu, Z. Zheng, S. Zhu, and P. Mohapatra, "Keeping context in mind: Automating mobile app access control with user interface inspection," in *2019 IEEE Conference on Computer Communications, INFOCOM 2019, Paris, France, April 29 - May 2*, 2019, pp. 2089–2097.

[10] W. Diao, X. Liu, Z. Li, and K. Zhang, "No pardon for the interruption: New inference attacks on android through interrupt timing analysis," in *Proceedings of the 2016 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, May 22-26*, 2016, pp. 414–432.

[11] Z. Xu, K. Bai, and S. Zhu, "Taplogger: inferring user inputs on smartphone touchscreens using on-board motion sensors," in *Proceedings of the Fifth ACM Conference on Security and Privacy in Wireless and Mobile Networks, WISEC 2012, Tucson, AZ, USA, April 16-18*, 2012, pp. 113–124.

[12] Q. A. Chen, Z. Qian, and Z. M. Mao, "Peeking into your app without actually seeing it: Ui state inference and novel android attacks," in *Proceedings of the 23rd USENIX Conference on Security Symposium, San Diego, CA, USA, August 20-22*, 2014, pp. 1037–1052.

[13] C. Ren, Y. Zhang, H. Xue, T. Wei, and P. Liu, "Towards discovering and understanding task hijacking in android," in *Proceedings of the 24th USENIX Conference on Security Symposium, Washington, D.C, USA, August 12-14*, 2015, pp. 945–959.

[14] M. Bentley, "Trojanized adware family abuses accessibility service to install whatever apps it wants." [Online]. Available: https://vhblog.lookout.com/blog/2015/11/19/shedun-trojanized-adware/

[15] J. Reardon, Á. Feal, P. Wijesekera, A. E. B. On, N. Vallina-Rodriguez, and S. Egelman, "50 ways to leak your data: An exploration of apps' circumvention of the android permissions system," in *Proceedings of the 28th USENIX Security Symposium (USENIX Security 19), Santa Clara, CA, USA, August 14-16*, 2019, pp. 603–620.

[16] C. Decker, "Bluewallet: The secure bitcoin wallet," in *Proceedings of the International Workshop on Security and Trust Management, Wroclaw, Poland, September 10-11*, 2014, pp. 105–112.

[17] R. Gennaro, S. Goldfeder, and A. Narayanan, "Threshold-optimal dsa/ecdsa signatures and an application to bitcoin wallet security," in *Proceedings of the International Conference on Applied Cryptography and Network Security (ACNS), Guildford, UK, June 19-22*, 2016, pp. 156–174.

[18] E. Androulaki, M. Karame, Ghassan O.and Roeschlin, T. Scherer, and S. Capkun, "Evaluating user privacy in bitcoin," in *Proceedings of the International Conference on Financial Cryptography and Data Security, Okinawa, Japan, April 1-5*, 2013, pp. 34–51.

[19] O. S. Okpara and G. Bekaroo, "Cam-wallet: Fingerprint-based authentication in m-wallets using embedded cameras," in *Proceedings of the 2017 IEEE International Conference on Environment and Electrical Engineering and 2017 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I CPS Europe), Milan, Italy, June 6-9*, 2017, pp. 1–5.

[20] D. He, S. Li, C. Li, S. Zhu, S. Chan, W. Min, and N. Guizani, "Security analysis of cryptocurrency wallets in android-based applications," *IEEE Network*, pp. 1–6, May 2020, doi:10.1109/MNET.011.2000025.