HORUS: A SECURITY ASSESSMENT FRAMEWORK FOR ANDROID CRYPTO WALLETS

MD SHAHAB UDDIN

A THESIS

IN

THE DEPARTMENT OF

CONCORDIA INSTITUTE FOR INFORMATION SYSTEMS ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF APPLIED SCIENCE

IN INFORMATION SYSTEMS SECURITY

CONCORDIA UNIVERSITY

MONTRÉAL, QUÉBEC, CANADA

SEPTEMBER 2021

© MD SHAHAB UDDIN, 2021

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Md Shahab Uddin**

Entitled: **Horus: A Security Assessment Framework for Android Crypto Wallets**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science
in Information Systems Security

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Dr. Jeremy Clark _____ Chair

Dr. Mohammad Mannan _____ Supervisor

Dr. Amr Youssef _____ Supervisor

Dr. Lianying Zhao _____ Supervisor

Dr. Jeremy Clark _____ Examiner

Dr. Ivan Pustogarov _____ Examiner

Approved by _____
Dr. Mohammad Mannan, Graduate Program Director

August 25, 2021 _____

Dr. Mourad Debbabi, Dean
Gina Cody School of Engineering and Computer Science

ABSTRACT

Horus: A Security Assessment Framework for Android Crypto Wallets

Md Shahab Uddin

Crypto wallet apps help cryptocurrency users to create, store, and manage keys, sign transactions, and keep track of funds. However, if these apps are not adequately protected, attackers can exploit security vulnerabilities in them to steal the private keys and gain ownership of the users' wallets. We develop a semi-automated security assessment framework, Horus[1], specifically designed to analyze crypto wallet Android apps. We perform semi-automated analysis on 311 crypto wallet apps and manually inspect the top 18 most popular wallet apps from the Google Play Store. Our analysis includes capturing runtime behavior, reverse-engineering the apps, and checking for security standards crucial for wallet apps (e.g., random number generation and private key confidentiality). We reveal several severe vulnerabilities, including, for example, storing plaintext key revealing information in 111 apps which can lead to losing wallet ownership, and storing past transaction information in 11 apps which may lead to user deanonymization.

[1]Horus is one of the ancient Egyptian deities. The Eye of Horus is an ancient Egyptian symbol of protection.

Acknowledgments

I would like to thank my supervisors, Dr. Mohammad Mannan, Dr. Amr Youssef and Dr. Lianying Zhao for their constant support and guidance throughout this project. Their continued support gave life to this project and made this research possible. I would also like to express my gratitude for their patience, motivation, enthusiasm, and immense knowledge. I am incredibly lucky to be able to work under the close guidance of my supervisors who inspired me with bright ideas, helpful comments, suggestions, and insights which have contributed to the improvement of this work.

I would also like to thank my peers at the Madiba Security Research Group for sharing their knowledge and experience and being there beside me on my rainy days. I learned a lot from everyone, especially, Tousif Osman and Xavier de Carné de Carnavalet. I feel honored to have worked with them. I feel lucky and grateful to be a part of this research group. Special thanks to all professors at CIISE. They all provided me with the opportunity to learn in a positive learning environment and made me more and more interested in all aspects of systems security.

I received substantial financial support from my supervisors and Concordia University. I am thankful to all for easing the financial burden while doing this research.

Lastly, I would like to thank my family and friends. This journey would not have been possible without their encouragement and support.

Contents

List of Figures

List of Tables

List of Acronyms

2FA Two Factor Authentication.

2FS Two-Factor Signatures.

ADB Android Debug Bridge.

AES Advanced Encryption Standard.

API Application Programming Interface.

APK Android Application Package.

ART Android Run-Time.

ATS Android Task Structure.

BCH Bitcoin Cash.

BIP Bitcoin Improvement Proposal.

CBC Cipher Block Chaining.

CPU Central Processing Unit.

DESede Triple Data Encryption Standard.

DEX Dalvik Executable.

DSA Digital Signature Algorithm.

EC-DSA Elliptic Curve Digital Signature Algorithm.

ECB Electronic Code Book.

ERC-20 Ethereum Request for Comments 20.

FORESHADOW FOREnSics of HArDware CryptOcurrency Wallets.

GPS Global Positioning System.

HD Hierarchical Deterministic.

HSM Hardware Security Module.

HTTP Hypertext Transfer Protocol.

ICC Inter-Component Communication.

IPC Inter Process Communication.

JAR Java ARchive.

JBOK Just-a-Bunch-of-Keys.

JRE Java Runtime Environment.

LTC Litecoin.

MITM Man In The Middle.

OWASP The Open Web Application Security Project.

P2P Peer-to-Peer.

P2PKH Pay-to-Public-Key-Hash.

P2SH Pay-to-Script-Hash.

PBE Password Based Encryption.

PBKDF2 Password-Based Key Derivation Function 2.

PKCS Public Key Cryptography Standards.

POC Proof of Concept.

PRNG Pseudo-Random number generator.

RAM Random Access Memory.

RPC Remote Procedure Call.

RSA Rivest–Shamir–Adleman.

SDK Software Development Kit.

SegWit Segregated Witness.

SSL Secure Socket Layer.

UC Universal Composition.

UI User Interface.

USDC USD Coin.

USDT Tether.

UX User Experience.

WIF Wallet Import Format.

XLM Stellar.

XML Extensible Markup Language.

XRP Ripple Digital Currency.

Chapter 1

Introduction

1.1 Overview

Bitcoin is the world's most popular cryptocurrency, its value has recently surpassed US$60,000 [64] and is getting wider adoption in businesses. Other cryptocurrencies like Ethereum, and Litecoin have established their footprints and going strong among the cryptocurrency users. The combined market cap of more than 6000 cryptocurrencies has reached a new high of US$1.24 trillion in 2021 [81]. Unfortunately, this massive growth of cryptocurrencies is also attracting malicious actors to find and exploit vulnerabilities in cryptocurrency related technologies. The cryptocurrency community has witnessed no less than 34 attacks, breaches, and scams in 2020 alone [34, 65], and attackers have stolen approximately US$4 billion [63] worth of assets from users.

Crypto wallets, being an ingrained part of the cryptocurrency ecosystem, also encounter attacks ranging from deanonymization to password cracking [35, 69, 74]. Due to their

importance, past work analyzed several crypto wallet apps. For example, He et al. [51] assessed critical attack surfaces in two wallet apps and demonstrated proof-of-concept attacks. Haigh et al. [50] analyzed the forensic artifacts of seven wallet apps, and designed a Trojan attack by repackaging a wallet app to steal user credentials; they chose to analyze non-HD (Hierarchical Deterministic [89]) wallets, although HD-wallets are gaining adoption in recent years and are preferred over non-HD wallets in terms of security and portability.

To perform a comprehensive and scalable analysis of current and popular wallet apps (the number of which is growing, currently in the range of hundreds), we developed a semi-automated test framework, Horus. Our framework combines both static and dynamic analysis of crypto wallet apps and can assess whether industry best practices are being followed in the app implementation. Horus has three major components: (1) *scraper module*, which can collect a large data set of specific categories of apps from the Google Play Store; (2) *static analysis module*, which looks for API pattern to determine if proper security standards are followed in the app implementation; and (3) *dynamic analysis module*, which searches for key revealing information stored in the app's artifacts (e.g., shared preference files, database files, and log files) on the device.

Using these components, we collect 311 wallet apps and analyze them for security risks. We conduct a manual inspection of the top 18 crypto wallet apps (part of 311 apps) in the Google Play Store to understand the apps' security practices and evaluate the security risk associated with them. We use popular open-source vulnerability analysis tools, such

as Androbugs[2] and Qark.[3] Syntax-based static analysis tools often fall short to determine vulnerabilities accurately [72]. We have noticed two issues with this approach, (1) a significant number of false-positive vulnerabilities and (2) the discovered vulnerabilities are generic and do not represent issues specifically applicable to wallet apps. To complement static analysis tools, we also conduct a dynamic analysis to better understand the apps' workflow, and explore a broader set of vulnerabilities, including plaintext key storage, and exported components. In the final step, we inspect the decompiled apps' code to verify our findings and understand the app's operational mechanism in greater detail. The main observation that follows from our analysis is that critical security standards applicable for wallet apps are missing in the app's binary, which indicates those standards are not implemented or have not been considered in the apps' development process. Additionally, the apps' key revealing information is handled insecurely (e.g., saved as plaintext or encrypted using a poorly chosen encryption configuration). Our framework can accurately identify critical issues in wallet apps, and our automated analysis results are consistent with the manual inspection results.

Several design components of our semi-automated framework take into consideration the implicit and varying nature of the wallet apps. Firstly, a wallet-import step varies significantly from one app to another due to the heterogeneous user interface, and the effort requires to make it a generalized automated step is non-trivial. To overcome this problem, we develop a semi-automated framework with a reduced-effort manual step (see Section 3.3). Secondly, most apps are still non-HD wallets, because app development is slower

[2]https://github.com/AndroBugs/AndroBugs_Framework
[3]https://github.com/linkedin/qark/

in comparison to the latest recommendation of the community, and do not support wallet portability features. We developed a separate workflow to analyze non-HD wallet apps (see Section 3.3.2). Lastly, some wallet apps are heavily obfuscated, which hinders our reverse engineering phase. To circumvent this problem, we focus on artifact analysis instead of the obfuscated code. This technique also enables us to be compatible with mobile apps developed using hybrid and cross-platform frameworks (e.g., PhoneGap, Flutter) along with native apps developed for Android and iOS. In this work, we keep our focus limited to analyzing Android apps.

1.2 Contributions

Our contributions can be summarized as follows:

1. We develop a semi-automated framework, Horus, to statically analyze wallet apps (fully automated), and perform dynamic analysis (with limited manual interactions).

2. We conduct an automated analysis of 311 crypto wallet apps on the Android platform. Additionally, we inspect the top 18 most popular crypto wallet apps, with combined downloads of 47M+ and in total 85M+ of 311 apps, in Google Play Store to understand the apps' security practices and evaluate the security risk associated with them.

3. We reveal that 111/311 of the apps store key revealing information in plaintext and 18 HD wallet apps store the encryption key without additional protections, i.e., without Android keystore or Android Hardware Security Module (HSM). Moreover, 11/311

apps store transaction information that can lead to deanonymization. Only 3/311 apps use HSM the secure storage solution to protect key revealing information.

4. We find that the Android user dictionary can be leveraged to derive the mnemonic phrase used to generate the private key. Only 20/311 apps implement custom keyboards to safeguard against this attack.

Most of the work presented in this thesis has been published in [83].

1.3 Thesis Organization

The rest of the thesis is organized as follows. In Chapter 2, we first present an overview of related works on desktop, web, hardware, and Android wallet apps, applicable attacks on wallet apps, and finally analysis frameworks for Android apps. In Chapter 3, we introduce our framework Horus and present the techniques we use to collect data sets, conduct static and dynamic analysis. In Chapter 4, we present experimental results of our analysis and discuss the impact of our findings. In Chapter 5, we discuss capabilities and limitations of our framework Horus and include best practices and recommendations for wallet app developers and users respectively. Finally, in Chapter 6, we present our concluding remarks and future work.

Chapter 2

Background

In this section, we provide brief descriptions of some wallet-related terminologies and our threat model.

Crypto wallet apps generally generate new addresses, store private keys securely, and help automate transactions. Some wallets can handle only one type of cryptocurrency (e.g., Bitcoin), and others can handle multiple types of cryptocurrencies. Furthermore, there are two types of wallets: Hierarchical Deterministic (HD) wallet [89], and non-HD wallet. HD wallets organize user accounts by one or more seed values and utilize open-source community-driven protocols to perform each operation, such as generating seeds and creating private keys. A HD wallet can deterministically generate all private-public key pairs used by the user, which ensures portability between multiple wallet implementations. Additionally, a HD wallet can generate practically unlimited number of key pairs for different transactions which ensures user's privacy and security by diversifying user's funds over multiple addresses. On the other hand, a non-HD wallet randomly generates keys (i.e., no

connection or hierarchy between the keys); such unrelated keys are also known as Just-a-Bunch-of-Keys (JBOK).

2.1 Bitcoin Improvement Proposal (BIP)

Several open-source community-driven protocols known as BIP [28] facilitate various crypto wallet functions. Each proposal is responsible for a specific goal, and the Bitcoin community can propose, rectify, establish, approve or reject proposals by consensus. As of March 13, 2021, there are 140 BIPs [28], but 3 BIPs are primarily relevant for HD-wallet apps. (1) *BIP 32* which defines a tree structure to populate public-private key pairs from a seed. The seed allows the wallet to be interchangeable with different implementations/devices, and implies the wallet does not need to be backed up often; just saving the seed is enough to recreate the tree structure of keys [89]. (2) *BIP 39* which defines both generating a mnemonic phrase and how to create a seed from the phrase, as compared to a hexadecimal random seed, a phrase is easier to remember and store for users. This proposal also defines a list of 2048 common English words to be selected for mnemonic phrases. The chosen words for a mnemonic phrase and their order are needed to regenerate the same seed afterward. The word list is also available in multiple languages [67]. (3) *BIP 44* which defines a syntax to enable a multi-account hierarchy of keys based on BIP 32. The syntax expresses purpose, coin type, account, address indices to generate proper keys [66]. The proposal defines how to generate any number of cryptocurrency-specific child keys. Figure 1 shows the life-cycle of a typical BIP.

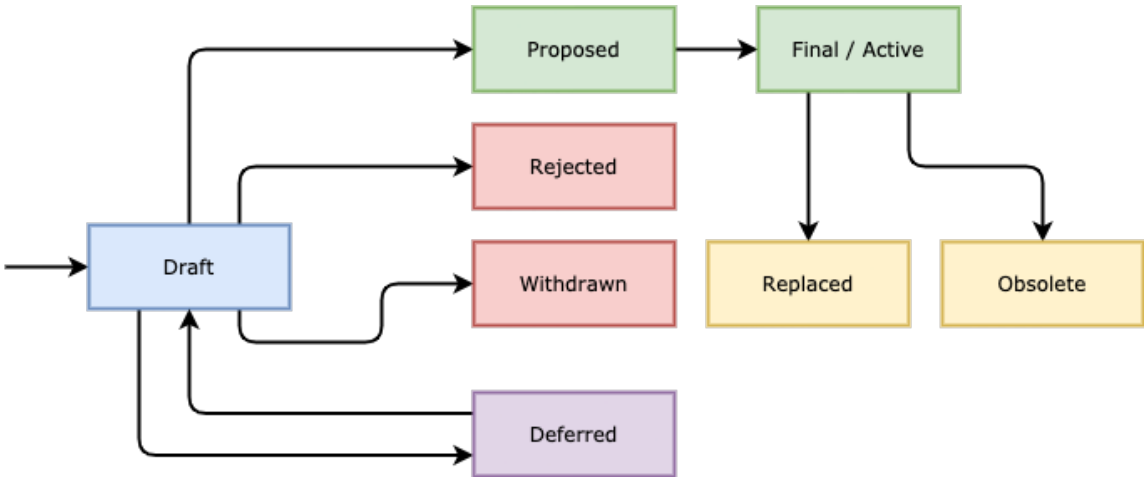

Figure 1: Life-cycle of a typical BIP.

2.2 Bitcoin Address Format

There are currently three Bitcoin address formats[4] available, each of which consists of between 26 to 35 alphanumeric characters. Generally, a wallet may not support all address formats.

Pay-to-Public-Key-Hash (P2PKH): Introduced in 2010, it is the hash of public key and is now known as the legacy address. The format supports standard transactions in Bitcoin, with one public key address transacting a value to another address. The vast majority of transactions on the Bitcoin blockchain are P2PKH. The address format starts with 1. However, it is not compatible with Segregated Witness (SegWit), but fortunately, Bitcoin can be sent from P2PKH to SegWit address with higher fees [32].

[4]https://allprivatekeys.com/bitcoin-address-format

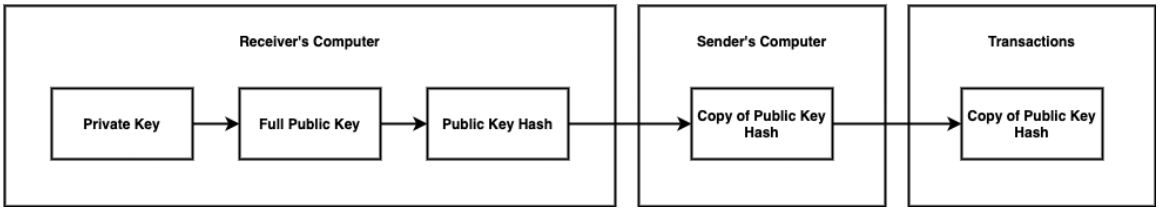

Figure 2: Workflow of a P2PKH transaction type.

Pay-to-Script-Hash (P2SH): The address begins with 3 and allows the transaction to be sent to a script hash which has a set of requirements that must be fulfilled before the value can be transacted. For example, the script could require multiple keys such as in a multi-signature transaction or need a password or any requirements one can build into the script. It offers more functionality than a legacy address, and it's most commonly employed for a multi-signature address where multiple digital signatures have to occur for the transaction to be authorized.

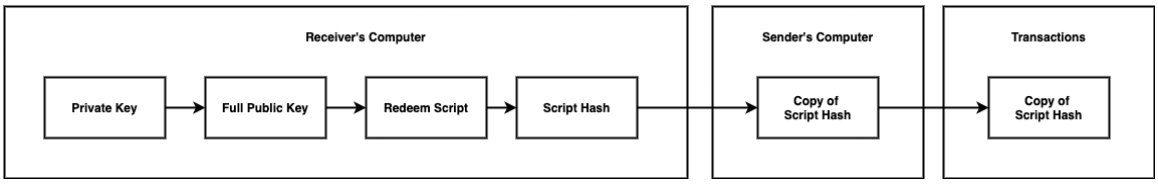

Figure 3: Workflow of a P2SH transaction type.

Bech32: It uses a 32-character set comprising the lower case alphabet and digits 2-7. It is more compact than other addresses to store in QR code. This address begins with bc1 and includes a 6-character checksum. The address is widely supported in terms of wallets, but not widely used. Less than 1% of Bitcoin is currently stored in this address format, but it is improving over time. More people are turning to Bech32 because there are lower transaction fees involved.

2.3 HD Wallet Generation Steps

A mnemonic phrase is generated using the standardized process defined in BIP 39. The most common phrase length is 12 and 24 words. In Android wallets, the 12-word length phrase is prevalent. First, a random sequence of 12/24 words is selected, providing 128-256 bits of entropy [24], see Figure 4. Then, the Password-Based Key Derivation Function 2 (PBKDF2) key derivation function is used to derive a 512-bit seed from the word sequence, see Figure 5. This seed can be used to deterministically generate an HD wallet [89]. An HD wallet root consists of a pair of master private key and a master chain code. Next, a master public key is generated from the master private key, see Figure 6. Both the master private key and the master chain code are used to generate child keys using BIP 44 [66]. Note that there is no way to verify the words and their order in a phrase. If the user adds/removes/scrambles the phrase's words, a new seed is generated, leading to some other wallet keys. The phrase, seed, master private/public keys, and master chain code are all key revealing information and should be protected with equal importance.

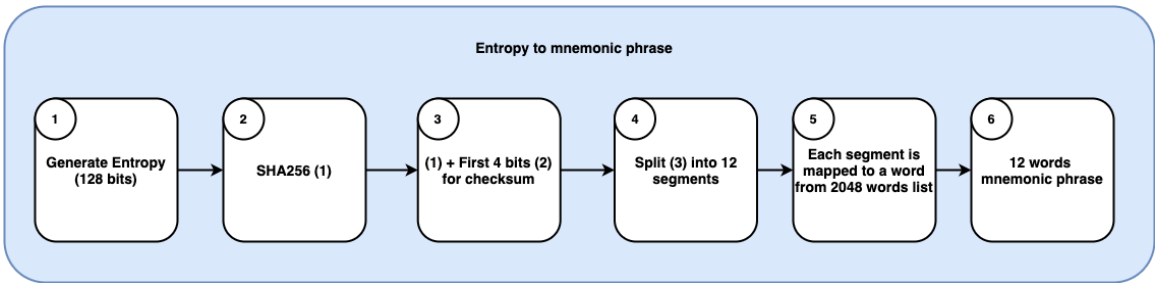

Figure 4: Generating entropy to mnemonic code steps.

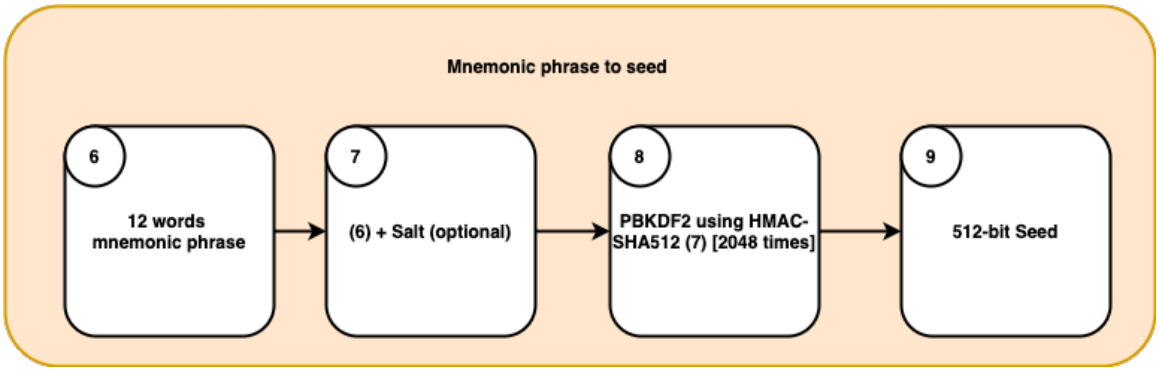

Figure 5: Mnemonic code to seed generation steps.

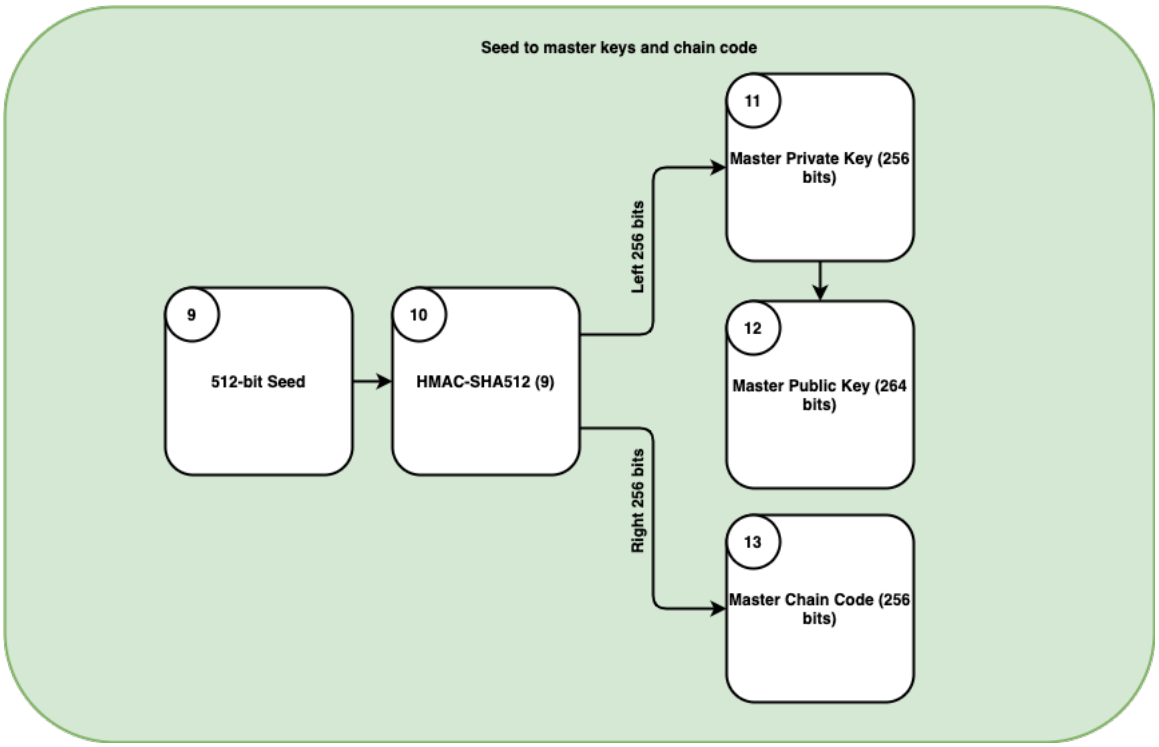

Figure 6: Seed to master private key and master chain code generation steps.

2.4 Wallet Import Format (WIF)

WIF formatted private keys improve the ease of typing and copy-pasting (include an error-checking mechanism to detect mistakes during typing or copy-pasting). WIF uses Base58

encoding (similar to Base64) and avoids ambiguous characters like 0, O, and I, l. WIF formatted keys are shorter than the real private keys. For example, Bitcoin's private key length is 64 bytes, and WIF formatted key length is 51 bytes. Base58 address also includes 4 bytes of SHA256 error-check code for automatically detecting copying errors. The error-check also reduces attacks where an address, that closely resembles an actual address, can be created by an attacker.

2.5 Wallet Transactions

A wallet transaction consists of two addresses and transferring coins between them. A transaction requires the receiver's public key and the sender's private key [29]. A sender can transfer any number of coins that the sender owns to the receiver's public key (or the receiver's address). The sender digitally signs the transaction with her private key to prove the transaction has been initiated and conducted by herself. Transactions happen in a network where both the sender and the receiver participate, known as the mainnet. There is also a parallel network named testnet [11], used for testing purposes, but testnet coins do not have any real value. Mainnet and testnet are two separate networks, and entities cannot transfer coins between them.

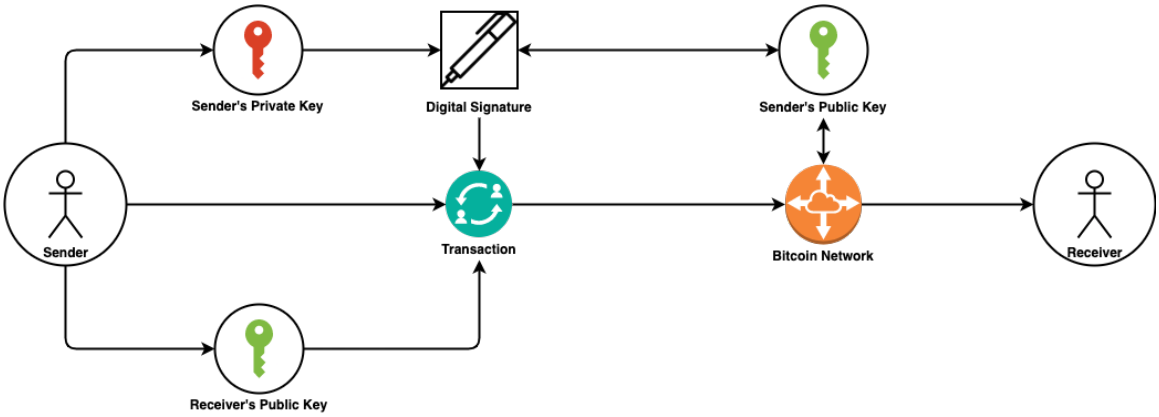

Figure 7: Bitcoin wallet transaction.

2.6 Threat Model

We assume the attacker can install a malicious app on the victim's device with the following capabilities (or a subset of those). The app can have virtual keyboard permission, or it is a keyboard app itself, which is set default by the user. This capability is required for user dictionary attack. The malicious app can take a screenshot of the device. We assume the device is not rooted by the user; however, a malicious app can successfully root the device. Root capability is required to capture key revealing information and to exploit cryptographic vulnerabilities. Attackers can have physical access to the unlocked device for a few minutes to exploit the allow backup attack. Note that not all these capabilities are required for all of our attacks.

2.7 Related Work

2.7.1 Desktop/Web Wallets Analysis

Several wallet app analyses have been carried out in recent years, which expose vulnerabilities and propose new defenses. Volety et al. [87] performed offline brute force dictionary attacks on the mnemonic phrase to gain access to two wallet apps. Guri et al. [48] infected a cold wallet with malicious code during the installation phase and get hold of the private keys. Further, Koerhuis et al. [56] conducted forensic analysis on two popular cryptocurrencies, Monero and Verge, in the desktop environment. They analyze the host machine's volatile memory, network traffic, hard disks and find critical artifacts like seed and plaintext passphrase. A similar study was carried out on Bitcoin by Zollner et al. [93]. Kaushal et al. [54] identified a variety of security threats to Bitcoin wallets including transaction suppression, theft, hijacking, etc. They recommend using only the Bitcoin core as it is safe against all feasible attacks. Gentilal et al. [43] designed a platform-specific optimized system to add flexibility and reliability to a Bitcoin wallet that is deployed in a non-trusted environment. They utilize an ARM extension named TrustZone to make the system resilient to side-channel and dictionary attacks.

2.7.2 Hardware Wallets Analysis

Hardware wallets for cryptocurrency are another category of wallets that highlights the functionality to prevent malware from hijacking digital wallets because the transaction

signing takes place on the hardware wallet and the private key never leaves the secure hardware wallet environment. However, studies show a substantial number of successful Man In The Middle (MITM) attacks on the hardware wallets [14]. Different producers make hardware wallets compatible with different interfaces, therefore the security depends on wallets integration, rather than on hardware wallet itself [80]. On the other hand, Pedro et al. [77] demonstrated two side-channel attacks on open source hardware wallets by extracting user PIN used for verification function and extracting the private key from the Elliptic Curve Digital Signature Algorithm (EC-DSA) scalar multiplication. Marcedone et al. [61] showed attacks on hardware wallets mounted by malicious hardware vendors and propose a Two-Factor Signatures (2FS) scheme that protects hardware wallets from such attacks. Thomas et al. [82] conducted memory analysis of cryptocurrency hardware wallet clients and develops a tool named FOREnSics of HArDware CryptOcurrency Wallets (FORESHADOW) to analyze memory dumps in Windows systems. Arapinis et al. [25] identified the properties and security parameters of Bitcoin wallets and defines a framework named Universal Composition (UC). The framework allows to capture wallet components' details and their interactions and they also define an attack under protocol deviation. Gkaniatsou et al. [45] demonstrated low-level protocol attacks on popular Ledger wallet communication and blamed the lack of well-defined security proprieties that Bitcoin wallets should conform with.

2.7.3 Android Wallets Analysis

Several studies also look into the security of Android wallet apps. He et al. [51] demonstrated two attack scenarios by capturing sensitive information from a device display using accessibility permissions and obtaining user input via USB debugging. This analysis is conducted on only two wallet apps (not among the top 50 wallets in the Google Play Store). Hu et al. [52] devised 3 proof-of-concept attacks targeting deanonymization, spamming, and violating Peer-to-Peer (P2P) protocol requirements of Bitcoin. Capturing clipboard values [57] also presents a significant risk to crypto wallet apps, e.g., when importing non-HD wallet keys from another app/device, or when copying mnemonic phrases. Haigh et al. [50] analyzed the forensic artifacts of seven wallet apps and develop a Trojan Proof of Concept (POC) by repackaging a wallet app that can steal the users' passwords. Gangwal et al. [12] used machine learning to identify a wallet app by tracing a user's network activity. Voskobojnikov et al. [88] analyzed 6,859 Play Store app reviews, related to User Experience (UX), of the top five mobile crypto wallet apps. The analysis suggests that both new and experienced users face significant struggles with the UX that might lead to dangerous errors or irreversible monetary losses. They also reveal misconceptions of the user such as some users believed that the transactions are free, reversible, and could be canceled anytime.

2.7.4 Android App Attacks Applicable to Wallets

Several studies identified and analyzed attacks on Android platform. The primary target of the attacks are not wallet apps but have impacts on wallet apps and can potentially be

leveraged to exploit wallet apps. User Interface (UI) deception attacks that include clickjacking, phishing, and activity hijacking [39] in Android, are generally applicable for any wallet app. Diao et al. [37] utilized interrupt information stored in /proc/interrupts and presents two attacks to infer unlock patterns and get the foreground app status. By leveraging the UI refresh pattern, an attacker can fingerprint the foreground app and can launch a phishing attack without requiring any permissions. Sai et al. [76] conducted static code analysis and network data analysis to discover common security vulnerabilities in Android apps. The experiment is based on The Open Web Application Security Project (OWASP) mobile top 10 vulnerabilities and takes banking and trading apps as a baseline. Reardon et al. [73] monitored runtime behavior and network traffic of apps to find out the side channels and covert channels that are used to leak sensitive data. Moreover, they describe how the permission model of Android can be exploited to gain access to sensitive data. Ren et al. [75] demonstrated task hijacking attacks utilizing Android's accessibility permissions and claims the attack can affect millions of Android apps. The exploitation uses Android Task Structure (ATS) to passively inject malicious UI in legit apps and displays a fake/-malicious UI to lure the user to enter their credentials. Chen et al. [33] proposed a new type of attack called UI state inference attack, which uses shared memory to infer the UI state changes and can be exploited to hijack sensitive information, such as user credentials. The attack can obtain sensitive information from the user's taken screenshots and does not require any permissions. Xu et al. [91] showed mobile phone's motion sensors, such as accelerometer, gyroscope, etc. can be exploited as the side channel to acquire data from the user. This study reveals an attacker can detect the location where the user taps on the

screen and can log the user's PIN/password.

2.7.5 Analysis Frameworks for Android Apps

Bergandano et al. [27] developed a hybrid analysis tool by following OWASP guidelines that analyze vulnerabilities in varying categories of apps (e.g., wallet, food, social). The work lacks manual inspection; thus, the tool's conclusion is unverifiable and focuses on generic vulnerabilities instead of taking into account any specific nature of the apps. DroidSafe [47] is a state-of-the-art Android static analysis tool that performs an app-level analysis to figure out potential information leakage in apps. It attempts to track both Intent and Remote Procedure Call (RPC) calls but lacks Inter-Component Communication (ICC), where two apps exchange data between them through a separate ICC. AndroidLeaks [44] is also a static analysis tool that states the ability to automatically determine private data leakage in Android apps, including phone information, GPS location, WiFi data, and audio recorded with a microphone. However, the tool fails to take context into account, preventing accurate analysis of many legitimate scenarios. To overcome the static analysis tool's false alarm issue, AppAudit [90] takes a dynamic analysis approach to simulate part of the program and perform checks at each program state. The goal of this tool is to reveal data leakage by apps and figure out the source of the data leakage. However, AppAudit still relies on static analysis to identify vulnerabilities and only uses dynamic analysis to confirm static analysis results and minimize false alarms. Ali-Gombe et al. [13] also took a hybrid approach called AspectDroid that detects suspicious behaviors in apps. In the static phase, apps have been repackaged with custom code inserted that performs logging and analytics

functionality. Then, dynamic analysis tracks and logs the runtime events of the app. The authors claim the tool can detect known malware apps correctly with an F-Score of 98%.

Contrary to prior studies, we explicitly focused on crypto wallet apps and discovered several new attack surfaces specifically applicable to Android wallets.

Chapter 3

Horus: Our Analysis Framework

The purpose of developing Horus is to automate the analysis process of wallet apps and discover issues in the apps' implementation. The idea and rationale behind building this framework and design decisions for different wallets are explained in this section. An overview of our evaluation methodology is presented in Figure 8. There are two modules for app analysis in Horus: *static analysis module* and *dynamic analysis module*.

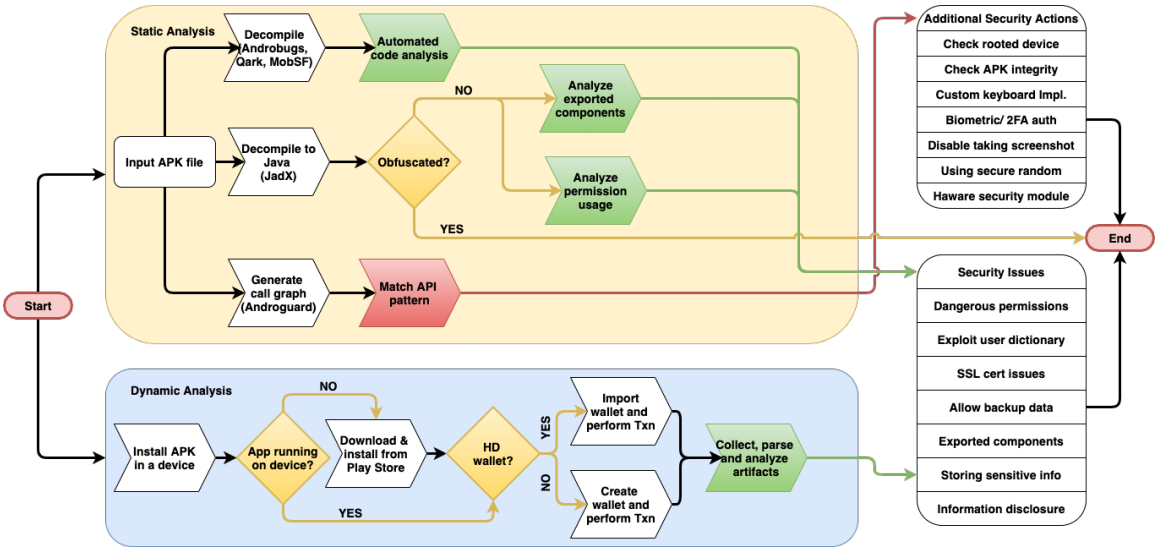

Figure 8: Overview of proposed framework.

3.1 Data-set Collection

Google Play Store search displays only the popular apps based on its search algorithm but does not provide a comprehensive list of all the apps matched with a search term [26, 71]. Conversely, a web search engine provides an exhaustive list of apps matched with the provided search term. We develop a specialized search engine scraper module based on a generic scraper tool, Search Engine Scraper,[5] for collecting a large set of apps. Our scraper module can search, parse results, remove duplicate app IDs, and download the Android Application Package (APK) files from Google Play Store automatically. We have used search term "site:play.google.com "bitcoin" "wallet"."

We use the search engine bing.com due to API restrictions in google.com (up to 100 pages, but multiple test runs from the same IP address may result in the IP being blocked before this limit is reached). We use a user agent value to appear as a desktop browser to the search engine, and introduce a random delay before scraping each new search result page to emulate normal usage and avoid any API restrictions.

We scraped 24,800 search results and found a total of 636 APK links, with 442 unique app IDs. We use PlaystoreDownloader[6] to download the latest version of the app from the play store. We encountered some exceptions during app download, including some apps are unavailable in our location (Canada), some apps are available only via the early access program, and some apps are incompatible with our device. Eventually, we downloaded 392 apps and found 82 apps were not wallet apps although the term "wallet" appears in their

[5]https://github.com/tasos-py/Search-Engines-Scraper
[6]https://github.com/ClaudiuGeorgiu/PlaystoreDownloader

Application ID	*Version*	*Downloads*	*HD?*	*Supported Coins*
asia.coins.mobile	3.5.22	5M+	No	BTC, XRP, ETH, BCH
co.bitx.android.wallet	7.6.0	5M+	No	BTC, ETH, XRP, USDC
co.mona.android	3.84.0	1M+	Yes	80+ coins
com.binance.dev	1.36.3	5M+	No	200+ coins
com.bitcoin.mwallet	6.9.10	1M+	Yes	BTC, BCH
com.breadwallet	4.7.0	1M+	Yes	BTC, XBT, BCH, ETH
com.coinomi.wallet	1.20.0	1M+	Yes	125+ coins
com.mycelium.wallet	3.8.6.1	1M+	Yes	BTC, ETH, ERC-20
com.paxful.wallet	1.7.1.534	1M+	No	BTC, USDT
com.polehin.android	3.4.9	1M+	No	100+ coins
com.unocoin.unocoinwallet	3.4.7	1M+	No	BTC, ETH, XRP, LTC
com.wallet.crypto.trustapp	1.26.5	5M+	Yes	50+ coins
com.xapo	5.3	1M+	No	BTC
de.schildbach.wallet	8.08	5M+	No	BTC
exodusmovement.exodus	21.1.28	1M+	Yes	50+ coins
org.toshi	23.3.357	1M+	Yes	100+ coins
piuk.blockchain.android	8.4.7	10M+	Yes	BTC, ETH, BCH, XLM
zebpay.Application	3.12.02	1M+	No	BTC, ETH, XRP, EOS

Table 1: Top 18 most downloaded wallet apps in Google Play Store (accessed: 2021-02-03). No indicates non-HD wallet app and Yes indicates HD wallet app.

description; such apps include Bitcoin key generator, currency exchange service, etc. We filtered out non-wallet apps and obtained a collection of 311 wallet apps. We shortlisted the most popular 18 wallet apps in Google Play Store for further manual inspection due to their high user base (with at least 1M+ downloads), see Table 1.

3.2 Static Analysis Module

The static analysis module of Horus looks for API patterns to determine if a particular functionality or security feature is implemented in the app. For instance, a wallet app must implement a custom keyboard; otherwise, it is vulnerable to user dictionary attacks; see

Section 4.2. We verify if the APIs required to implement a custom keyboard in Android are called in the app. If the API calls are found, the functionality is assumed to be implemented in the app.

To determine the API calls in an app, Horus takes an APK file as input and constructs a call graph of the app using Androguard.[7] A call graph contains the class name, method name, descriptors, and access flags. Each node in the call graph is a method, and the actual calls with arguments are denoted by edges. For root detection, the app checks the execution of su command [55], or the existence of a list of root enabler apps [38] on the device. The call, Runtime.exec() is used to check the existence of su binary and PackageManager.getPackageInfo() is used to check the existence of a root enabler app. The API calls in the app's call graph indicate the app is checking whether the device is rooted. All the API calls required to implement a functionality constructs an API signature. Table 2 depicts the API signatures we use to determine the existence of the security standard in an app. Below, we discuss several security standards that we verify using our static module.

3.2.1 Secure Random Generator Usage

A secure random number generator is crucial for crypto wallet apps. According to BIP 39 [67], to create a mnemonic phrase, the client must use an entropy of length 128-256 bits.

[7] https://github.com/androguard/androguard

Type	*API Signature*	*Usage*
Root Detection	Runtime.exec()	Execute runtime command, e.g. su
	PackageManager.getPackageInfo()	Get installed app's information
	Os.stat()	System call for runtime command execution
	Os.access()	System call to query installed app
Integrity Check	PackageManager.getPackageInfo()	Get installed app's information
	Context.getPackageCodePath()	App installer file path
	ZipFile.init()	Execute operation on installer file
	RandomAccessFile.init()	Read, write in system file
Custom Keyboard	KeyboardView.setKeyboard()	Replaces default keyboard app
	OnKeyboardActionListener.onKey()	Listeners for input key
	InputMethodService.onCreateInputView()	Callback when the keyboard view is created
	InputConnection.commitText()	Commit the user input to app
	InputMethod	Handles keyboard type depending on input field
Biometric Authentication	BiometricManager	Provides biometric utilities
	BiometricPrompt	Handles biometric authentication
	FingerprintManager	Defines types of authentication (Deprecated)
	BiometricService	Updates system server for fingerprint
	FingerprintService	Updates system server for fingerprint (Deprecated)
Screenshots Disabled	Windows.setFlags()	Uses for full screen access
	View.setDrawingCacheEnabled()	Access to the current view displayed
Hardware module	KeyStore.getInstance()	Returns a keystore object of specified type
	KeyGenParameterSpec.Builder.isStrongBoxBacked()	Requesting Android to use hardware module
	StrongBoxUnavailableException	Exception if the hardware module is absence
Random Generator	SecureRandom	Cryptographically strong random number generator
Dynamic Code Loading	DexClassLoader	Loads classes from JAR or APK file
	dex2oat	Utility program used to install and update apps
	libart	An so file related to ART
	DexFile	Loads DEX files (Deprecated)
	PathClassLoader	A class loader implementation from local file system

Table 2: API signatures to check for feature existence.; the *Type* column denotes the security standard type, the *API Signature* column points to the API calls and individual class names required to implement the security standard, and the *Usage* column indicates the use case of each API call.

A seed is generated from the mnemonic phrase and is used to create the master private/public keys. If the random number generator is predictable, it affects the key generation objective. In Android, /dev/urandom [85] is used to generate seed for SecureRandom [23], which is the recommended API to generate a cryptographically strong random number.

3.2.2 Implementing Custom Keyboard

Once a wallet app generates a mnemonic phrase, it asks the user to enter the mnemonic phrase for verification. To import an existing wallet into a new app, the user also needs to enter the mnemonic phrase via keyboard. If the user uses a third-party keyboard app, it can capture all user inputs [20], including the mnemonic phrase. Additionally, the app is vulnerable to user dictionary attack. A wallet app should have a custom keyboard triggered while taking any key revealing input and possibly randomize the keyboard's key location [59].

3.2.3 Disabling Screenshot-taking

This is another critical risk avoidance feature for wallet apps. Any malicious app with the capability of taking a screenshot can capture the screen content during the wallet import phase. Moreover, a user may take a screenshot of the mnemonic phrase to save it as a backup. This image is saved in the gallery, and any app with reading storage permission can access the file. Crypto wallet apps must call Android API to disable the screenshot-taking feature for sensitive screens.

3.2.4 Two Factor Authentication (2FA)

2FA or biometric authentication, if supported by the device, should be implemented before performing any sensitive operation. We observe a significant number of wallet apps do not require user registration. An unauthorized user can perform a transaction with a few minutes of physical access to the device, assuming it is unlocked. Some apps require the user to set a PIN code and ask for the PIN code when confirming a transaction. However, a PIN code can be as short as a 4-digits number, which can be brute-forced or seen by shoulder surfing. A 2FA should be incorporated in each wallet app as second-layer protection.

3.2.5 Integrity Check

App's signature verification to check integrity ensures that the app has not been tampered with and installed from a legit source (e.g., Google Play Store). Integrity can also be ensured by calculating the hash of the installed APK file and comparing it with the hash of the authenticated APK file. The solution is not foolproof and can be bypassed in a repackaged app. However, integrity checking is a widely adopted practice in financial apps and is considered a self-defense mechanism. The wallet app must perform integrity checks as another layer of security before starting to operate.

3.2.6 Root Detection

An app can be made more secure by implementing 2FA or strong encryption; however, no security on the app end works if the device itself is compromised. Malware apps can

have root providers included in the binary and may gain root privilege without the user's consent [92]. This privilege can be exploited in many ways, from monitoring activities of other apps to sending key revealing information to a malicious back-end.

3.2.7 Hardware Security Module

Recent Android smartphones have a separate computing environment, which provides additional security to keep key revealing information safe, called hardware security module [19]. The key revealing information is stored in a secure enclave that is also protected in a rooted device and safe against brute force attack by utilizing rate limiting. The solution is not foolproof; there is still room for information disclosure if the device contains a malicious keyboard app or a clipboard listener. Nevertheless, the hardware security module provides substantial improvements over any other storage solutions and should be used by wallet apps to secure key revealing information. To determine the existence of HSM in the device and to perform operations on HSM, Android provides high level APIs (Table 2) that abstract out the different hardware module implementation provided by vendor.

3.2.8 Dynamic Code Loading

Dynamic code loading defines an act of the app to load and execute additional code provided from the server after the app is installed. This additional piece of code is not bundled with the app itself and can be downloaded at any later point in time based on the user's device capability and user's behavior. Dynamic code loading does not require user interaction and is capable to disregard the operating system's code loading authority. This can

present a threat to the user's privacy and security and allows the developer to update the app without the user's consent. This capability facilitates injecting malicious code that can be exploited by an attacker [58].

3.2.9 Manual Inspection

For in-depth inspection of the top 18 wallet apps, we use four state-of-the-art analysis tools, AndroBugs, Qark, MobSF[8] and Cryptoguard [70]. The tools look for generic vulnerabilities by following a defined set of rules and patterns and do not require the app to run. The first three tools start with decompiling the app code and looking for several vulnerabilities, including runtime command execution, Secure Socket Layer (SSL) certificate verification, cryptographic API misuses, and webview vulnerabilities. The first three tools mark the found vulnerabilities with different severity labels (e.g., Critical, Warning, Info) and we consider only the critical vulnerabilities. The last tool, Cryptoguard, exposes cryptographic API misuses (e.g., predictable keys, constant passwords, vulnerable certificate verification) by implementing forward and backward program slicing techniques. Cryptoguard defines 16 categories of vulnerabilities for Android apps and looks for those in APK files. We observe that Cryptoguard failed to generate output for some of the apps (e.g., co.mona.android, com.wallet.crypto.trustapp, piuk.blockchain.android, etc.). The reason for the failure is insufficient memory for the Java Runtime Environment (JRE). To overcome this issue, we allocate 32GB RAM for the process to run and adopt some techniques,

[8]https://github.com/MobSF/Mobile-Security-Framework-MobSF

such as, forcefully calling the garbage collector after each category of vulnerability analysis of an app and restarting the process to release the allocated memory used for each app analysis before starting the analysis of a new app. This allows us to minimize the total number of failed analyses, however, the issue still persists and requires additional technical investigation which is beyond our scope.

We use reverse engineering to discover vulnerabilities in the app code by decompiling the APK file. Reverse engineering reveals permissions usage in the app for malicious purposes, logic bombs, Trojan code, etc. We also use reverse engineering to verify the findings of analysis tools on the source-code level to understand the impact of the vulnerabilities. Sometimes the app code is obfuscated by renaming code components, such as folders, classes, variables, into a shorter, unintelligible name [17]. It is difficult to get out any meaningful information from the obfuscated code; we skip the app in such cases. We use Apktool[9] and JD-GUI[10] for our reverse engineering step. Apktool can decode resources from the Android APK file, and JD-GUI is a Java decompiler and code browser.

3.3 Dynamic Analysis Module

In the dynamic analysis module, we look for key revealing information in apps' internal file structures. If the device is rooted or the target app allows backup, malicious apps or actors can capture the target app's internal files. If the internal files contain key revealing

[9]https://ibotpeaches.github.io/Apktool/
[10]https://github.com/skylot/jadx

information (e.g., plaintext private key), a malicious actor can easily access such information. In general, the master private key is the most critical information that needs to be protected by wallet apps. Other such critical items include the mnemonic phrase, seed, and master chain code—all of which should be protected with equal importance as of the master private key. Note that a mnemonic phrase is used to generate a seed value, which is used to generate its corresponding private key. Similarly, the master public key and any of the child's private key is enough to recreate the master private key [49]. Therefore, it is imperative to secure all key revealing items along with the master private key. Our goal is to seek answers to the following questions: Are wallet apps storing the above-mentioned key revealing information in plaintext on the device? If encrypted, which encryption algorithm is used? Can we identify the encryption key used to perform the encryption, and if yes, where is the encryption key stored?

3.3.1 HD Wallet Analysis Workflow

A HD wallet can recreate the hierarchical tree of keys from a mnemonic phrase. In our HD wallet workflow, we maintain four lists as follows. (1) *Key revealing information*: A list of secret information that can be used to regenerate the master private key. In addition to the master private key itself, this list includes mnemonic phrase, seed, seed hex value, BIP 32 private key, and master chain code. (2) *Candidate encryption key*: Key revealing information should be encrypted before storing on the device. *Candidate encryption key* is a list of all possible encryption keys that wallet apps can use to encrypt key revealing information. We find traces that the encryption keys are stored in the app's internal files.

Using Horus, we can verify how widespread the practice is to store encryption keys locally. (3) *Cipher key revealing information*: Each item of the *key revealing information* list is encrypted and the resulting ciphertext is encoded using Base64 and then appended to this list. (4) *Search term*: A combined list of *key revealing information* and *cipher key revealing information*. We use this list of items as a search term and look for each item of this list in the wallet's artifact.

To start the dynamic analysis, we install the target app on a rooted device. We generate a mnemonic phrase and import the same phrase in all HD wallet apps. We use a fixed email, username, PIN, password, and phone number in all the apps for account creation and verification as needed by the app. We append this fixed information in the *candidate encryption key* list as potential encryption keys.

Using a mnemonic phrase, Horus gets *key revealing information* list, and by parsing internal files, Horus get *candidate encryption key* list. Horus iterates through the *key revealing information* list and applies encryption algorithms available in the Android platform to encrypt each of the items in *key revealing information* by using items from *candidate encryption key* list as the encryption key. One such operation is as follows, we take the first element of *key revealing information*, encrypt it with AES using the first element of *candidate encryption key*, and add the resulting ciphertext in *search term*. We then repeat the same operation with the next element of *candidate encryption key*, and so on. When done, we use a different encryption algorithm (e.g., Blowfish) and go through the same list of *candidate encryption key* and repeat the process. All the *key revealing information* is also appended as-is to *search term* list because the *key revealing information* can also be

found in internal files without any encryption. Horus reads all the app's internal files to find traces of the elements in *search term*. If any of the *search term* elements are found, then the wallet app is exploitable if the internal files are exposed.

Horus generates a mnemonic phrase used for all HD wallet apps throughout the analysis. Horus starts a tcpdump[11] session to capture all network requests and clear the logcat buffer for capturing a new session. At this stage, we import the mnemonic phrase in the app. Then, Horus takes the app ID as input and pulls all internal files, tcpdump generated network dump files, from the connected emulator/device, and reads the files sequentially.

Horus identifies common file extensions (e.g., xml, db, pcap) and uses an appropriate parser to extract the file content. In case of an XML file, Horus parses the XML file and reads the content and appends all string values in *candidate encryption key*. For the SQLite database file, we develop a parser that lists all tables in the database and reads all the values in the tables and appends them in the same list. We get a pcap file from tcpdump containing network request logs, and we use a Python library Scapy[12] to parse the pcap file and extract the content. Horus can also parse log files and take fixed email, username, PIN, and password used as input. All the parsed content and input are considered potential encryption keys and listed in *candidate encryption key* list.

Horus reads all internal artifacts again and runs a fuzzy search, using fuzzywuzzy,[13] for the items listed in *search term*. We record the search result whether *search term* items are found and whether in plaintext or encrypted form. Note that all *candidate encryption*

[11] https://www.tcpdump.org/
[12] https://scapy.net/
[13] https://github.com/seatgeek/fuzzywuzzy

key items cannot be used directly as encryption keys. AES requires blocks of 16 bytes key, whereas a PIN is a four-digit number. Thus, a PIN cannot be used as a key. We use four common hashing algorithms (MD5, SHA1, SHA256, and SHA512) to generate a digest for each *candidate encryption key* and use the digest instead as an encryption key. Throughout our reverse engineering step, we observed that the hashing technique to convert a non-suitable key into a proper encryption key is followed in many wallet apps.

3.3.2 Non-HD Wallet Analysis Workflow

A non-HD wallet generates a list of public-private key pairs, and there is no relationship among the keys. Non-HD wallets manage many keys; each public-private key pairs are used for only one transaction. The downside of this approach is that the user needs to take backup regularly, ideally after each transaction. This approach is not convenient and error-prone. Additionally, there is a risk of key exposure if the backup is not handled with caution. In Horus, non-HD wallet workflow is different from HD wallet workflow because, in an HD wallet, keys can be generated predictably with a known mnemonic phrase, but there is no relation among the generated keys in non-HD wallet. Therefore, we look for key patterns in the internal files in the non-HD wallet workflow. For instance, Bitcoin public addresses start with character 1 or 3 [78], are 34 characters long, and formatted as Base58; Ethereum public addresses start with 0x and are 42 characters long. We consider different key formats as well. For example, Bitcoin private keys can be in standard 256-bit hex format (64 bytes long), or WIF format [60] (51 bytes long) and start with 5. The

prerequisite steps of performing dynamic analysis on HD wallets are applicable for non-HD wallets except importing the wallet. Instead of importing the wallet, the keys are generated using the app itself. Different apps of course generate different keys; however, the key format is identical. We incorporate a pattern matching regex to look for Bitcoin public/private addresses and their various derivatives [40] in the app's internal files.

3.3.3 Transactions Workflow

For both HD and non-HD wallets, we make a small transaction [29] of a fixed amount to a pre-defined address. Horus looks for the predefined receiver's address and the fixed amount value in the wallet app's internal files. After a transaction is completed, the transaction history should not be saved in the device. If the receiver's address and the transaction value are present in the app's internal files, it stores past transactions and can be abused for deanonymization if the app's internal files are exposed.

To emulate wallet transactions, we use Bitcoin testnet [11]. Since not all wallet apps support testnet transactions, we make transactions with only testnet compatible wallet apps. To collect testnet coins, we use coinfaucet[14] and mempool,[15] two freely available services to distribute testnet coins.

3.3.4 Manual Inspection

For in-depth analysis of the top 18 wallet apps, we monitor the apps' workflow and response based on the app's interaction. Our goal is to understand the app's workflow and its process

[14] https://coinfaucet.eu/en/btc-testnet/
[15] https://testnet-faucet.mempool.co/

to generate and store key revealing information. We observe changes in the app's internal files (e.g., shared preference files, database files, log files, file IO) based on the activity we perform using the app.

An Android app consists of 4 components: Activity, Service, Broadcast Receiver, and Content Provider. Android enforces a sandbox mechanism to protect the components, where no app gains access to other app's components by default. However, an app can export its components and let other apps access the components. If a component, such as a service, is exported and not protected with permissions, then any app can start and bind to the service. Any app on the device can invoke all the exported components in the target app. We manually verify if it is possible to send a crafted intent from any other app to activate the exported components in the target app and make it perform the malicious task.

We use three state-of-the-art tools, Drozer[16], Frida[17] and Crylogger[68]. We use Drozer to list all exported components, universally accessible URIs using which any other app can ask for key revealing information from the target app and SQL Injection attack surface in the app. We use Frida to monitor critical operations in the app, e.g., database operations, file IO, and method trace to determine passing arguments and the return value of a method. We also use Frida to trace app logs, bypass SSL pinning, and bypass root detection. We use an additional wrapper tool, House[18] over Frida for ease of use. Both Drozer and Frida are used for monitoring and intercepting the app workflow. We use Crylogger to dynamically monitor cryptographic API usage and their parameters to detect crypto rules violation.

[16]https://github.com/fsecurelabs/drozer/
[17]https://frida.re/
[18]https://github.com/nccgroup/house

This tool uses Android Monkey[19] to generate random events in the app UI and capture logs of crypto API usage. There are 26 defined rules in the tool and each app is verified against these rules. Some rules require two executions of the app and the tool checks if some values are present or being used on multiple executions of the app. We observe that 31 apps in our data set did not produce any output because of the unavailability of libraries in the emulator environment or the unsupported architecture of the emulator. We configure Monkey to generate 30,000 events on each app, which is an optimal balance between sufficient events and the time required to capture an app's logs. However, Monkey cannot perform complex operations, such as importing keys or login, and generated events are random and therefore not comprehensive. To compensate for this issue, we modify the flow of the tool and manually navigate all the available screens of top apps to trigger all available functionalities of the apps and let the tool capture the crypto API usage logs.

[19] https://developer.android.com/studio/test/monkey

Chapter 4

Experimental Results

We use the LDPlayer[20] emulator, running Android 7.1.2 for all automated analysis, and Alcatel 5041C, running Android 8.1.0 for manual experiments. We use LDPlayer for its faster execution. The two different Android versions we use to cover a large part of contemporary Android phone users. Dynamic analysis requires a rooted device and we root the device using Magisk.[21] We analyze 311 wallet apps using Horus and manually inspect the top 18 most popular apps. In this section, we present some of our main findings and corresponding security risks.

4.1 Storing Key Revealing Information

Our dynamic analysis identifies 239 apps (77%) as non-HD wallets, 71 apps (23%) as HD wallets, by verifying whether the wallet has the functionality to create or import a wallet

[20] https://www.ldplayer.net/
[21] https://github.com/topjohnwu/Magisk

using a mnemonic phrase. In total, 111 apps (87 non-HD wallets and 24 HD wallets) store key revealing information in plaintext. In 47/71 HD wallet apps use encryption to store key revealing information; however, 18 of those apps store the encryption key without additional protections. In most cases, where we found an encryption key, the key is located in the SharedPreference file. In other cases, the key is located in a readable internal file, or it is a user-provided PIN (e.g., 4-6 digits). Among the top 18 apps, 3 apps store a key revealing information in plaintext, and 4 apps store key revealing information encrypted with a known encryption key from our list of *candidate encryption key*. The secure storage solution in Android is HSM, but only 3/311 apps are using HSM. If HSM is not available, then Android keystore should be used, which provides the best available solution provided the Android OS itself is not compromised. The user-provided PIN should not be used as the encryption key because it is brute-forceable. 11 of 311 apps store transaction information on the device, leading to deanonymization if the internal files are exposed. The transaction storage feature is advertised as a usability feature of the wallet so that the user can view all previous transactions. However, the transactions can be obtained on-demand without storing the information on the device. To violet the anonymity of the collected transaction information, an attacker is required to know the identity of the owner of the device.

4.2 User Dictionary Attack

On all Android devices, the keyboard app uses a user dictionary (database of words, locales, and frequency count) for predictive text input. In wallet apps, the mnemonic phrase

Application ID	*Plaintext Sensitive Info*	*Encrypted Sensitive Info*	*Encryption Algorithm*	*Encryption Key Location*	*Saved Transaction Info*
asia.coins.mobile	✗				
co.bitx.android.wallet		✗	AES	SharedPref	
co.mona.android					
com.binance.dev					
com.bitcoin.mwallet					
com.breadwallet		✗	AES	SharedPref	
com.coinomi.wallet					
com.mycelium.wallet		✗	AES	SharedPref	✗
com.paxful.wallet	✗				
com.polehin.android	✗				
com.unocoin.unocoinwallet					
com.wallet.crypto.trustapp					
com.xapo					
de.schildbach.wallet					
exodusmovement.exodus					
org.toshi					✗
piuk.blockchain.android		✗	AES	PIN	
zebpay.Application					
Total	111	38	AES: 12, Blowfish: 1		11

Table 3: Horus dynamic module analysis summary. ***Plaintext Sensitive Info*** column indicates the presence of plaintext sensitive information in wallet artifacts; ***Encrypted Sensitive Info*** column indicates the sensitive information is encrypted, however the encryption key exist in the device; ***Saved Transaction Info*** column indicates the transactions information is saved in artifacts, blank cell indicates absence of information. Last row provides the total number for all 311 apps.

contains common English words, and most wallet apps take the mnemonic phrase input from the user using the default keyboard. The information regarding the words in the mnemonic phrase is saved in user dictionary [36]. This dictionary can be abused to predict the mnemonic phrase [53] by extracting frequency information of typed words. Note that the attacker app requires virtual keyboard permission to access the dictionary, and in general, input editor and spellchecker apps ask for this permission. Multiple apps can have virtual keyboard permission on a device. Only 20 apps out of 311 apps (6%) implement

custom keyboards to defend against this attack. To exploit this vulnerability, the existing dictionary data is required to be reset, or a snapshot of current dictionary state is preserved. A malicious app having virtual keyboard permission can reset the database. Then the malicious app inserts the common 2048 English words with the frequency 0 in the database. From that point onward, if the user types any word from the 2048 English words using the default keyboard, the dictionary frequency column is updated. The maximum value of the frequency column is 255 and when a word usage is reached to the maximum frequency value, then the frequency column will is fixed with 255, no matter how many times that word is used at any later point. Also, the word length has to be at least 3 characters, otherwise, the frequency value doesn't get updated, even if the word exists in the database. However, this update mechanism is not real-time and requires the target app to be closed or go in the background. Using the updated frequency column an attacker can predict the typed words by comparing the frequency value. The attacker can only get the list of typed words but not the order, which is crucial to generate the correct master key. The number of combinations of a 12-word mnemonic phrase is not high (500 million) and all possible combinations can be brute-forced within a reasonable time. To determine the feasibility of the brute force attack, we experiment to generate all possible combinations of words in a mnemonic phrase. We use Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz with 32 GB RAM and running Windows 10 operating system, and it requires less than an hour (3540.96s) to enumerate all possible combinations and generate the corresponding master key and public key. There are several API services (e.g., Blockchain API[22]) to verify the account balance

[22]https://www.blockchain.com/api

of a particular public key.

4.3 Enabling Allow Backup

This is an attribute declared in the AndroidManifest.xml file, and it is true by default. It denotes the app data is backed up upon the app's uninstallation and is restored upon re-install [18]. When enabled, the app data can be backed up using the Android Debug Bridge (ADB) command. It enables an attacker to extract an app's internal files from a non-rooted device within a few minutes of physical access. Open-source tools such as Android-Backup-Toolkit[23] can be used to extract the backup and gain access to internal files. We observe 134/311 apps have allow backup attribute enabled in the AndroidManifest.xml file.

4.4 Dangerous Permissions

In the Android ecosystem, some permissions are considered dangerous [21] and require the user's explicit consent before being authorized. Most wallet apps require some common permissions for their functionality (e.g., WRITE_EXTERNAL_STORAGE, READ_-EXTERNAL_STORAGE, GET_ACCOUNTS, CAMERA). However, some apps ask for certain privacy-sensitive permissions, such as contact list and record audio, which appear to be non-essential for the app; see Table 4 for such permissions. Each permission has a constant value associated with it, such as android.permission.RECORD_AUDIO for audio

[23] https://sourceforge.net/projects/android-backup-toolkit/

recording, which is used to check, request, and verify permission from the Android Software Development Kit (SDK). The constant values of the permissions are stored as a static variable in Manifest.permission class [21]. The presence of the static variable or the associated constant value of a particular permission in the app's codebase confirms the usage of the permission.

Application ID	***Read Contacts***	***Access Coarse Location***	***Access Fine Location***	***Read Profile***	***Read Phone State***	***Get Tasks***	***Request Install Packages***	***Record Audio***
asia.coins.mobile	✗	✗	✗					
co.bitx.android.wallet	✗			✗				✗
co.mona.android	✗		✗					
com.binance.dev		✗	✗					
com.bitcoin.mwallet	✗	✗	✗					
com.breadwallet		✗	✗					
com.coinomi.wallet								
com.paxful.wallet					✗	✗		
com.polehin.android								
com.unocoin.unocoinwallet	✗	✗	✗					
com.wallet.crypto.trustapp							✗	
com.xapo	✗	✗	✗	✗	✗	✗		
de.schildbach.wallet								
exodusmovement.exodus								
com.mycelium.wallet		✗						
org.toshi								
piuk.blockchain.android								✗
zebpay.Application						✗		
Total	47	75	81	8	84	36	28	40

Table 4: Unnecessary dangerous permissions usage. Last row provides the total number for all 311 apps. ✗ indicates the app declares the permission requirement in the AndroidManifest file; *Read Profile* allows an app to read the user's personal profile data; *Read Phone State* allows the app to access the device's phone features; *Get Tasks* allows the app to retrieve information about currently and recently running tasks; and *Request Install Packages* allows app to install additional packages on the device.

4.5 Root Exploitation

In the Android ecosystem, root exploitation is well-known [84]. There are legitimate Android apps available in the Google Play Store that facilitate the rooting of phones, referred to as root providers or one-click root apps. In 2016, 85 million devices downloaded such root provider apps, and the devices are soft-rootable [42]. In wallet apps, we find 70 apps out of 311 are checking if the device is rooted before starting its operation; see Table 5. However, none of the apps terminates upon root detection; instead, the app displays a non-blocking alert and lets the user continue using the app.

Application ID	***Root Detection***	***Integrity Verify***	***Screenshot Disabled***	***Bio-metric***	***Custom Keyboard***	***Secure Random***	***Dynamic Code Loading***	***Hardware Security Module***
asia.coins.mobile	✓		✓			✓		
co.bitx.android.wallet	✓			✓		✓		
co.mona.android			✓			✓		
com.binance.dev	✓		✓		✓	✓	✓	
com.bitcoin.mwallet	✓	✓	✓		✓	✓		
com.breadwallet		✓			✓	✓		
com.coinomi.wallet	✓	✓				✓		✓
com.mycelium.wallet			✓		✓	✓		
com.paxful.wallet				✓		✓		
com.polehin.android			✓	✓		✓		
com.unocoin.unocoinwallet		✓				✓		
com.wallet.crypto.trustapp	✓		✓		✓	✓		
com.xapo	✓			✓	✓	✓		✓
de.schildbach.wallet						✓		
exodusmovement.exodus	✓		✓		✓	✓		
org.toshi			✓			✓		
piuk.blockchain.android	✓		✓		✓	✓		
zebpay.Applicaiton						✓		✓
Total	71	33	45	46	21	293	10	10

Table 5: Horus static module analysis summary. Here, ✓ indicates the existence of security standard implementation in the app and a blank cell indicates the absence of it. Last row provides the total number for all 311 apps.

4.6 Strandhogg Attack

This attack [10] is applicable when a malicious app that targets a wallet app opens up before the wallet app. When the user taps on the wallet app, the malicious app opens up instead. The malicious app can mimic the wallet app's UI and ask for its PIN, mnemonic phrase, etc. No permission is required for the malicious app, and Android versions 8.0-9.0 are vulnerable to this attack.

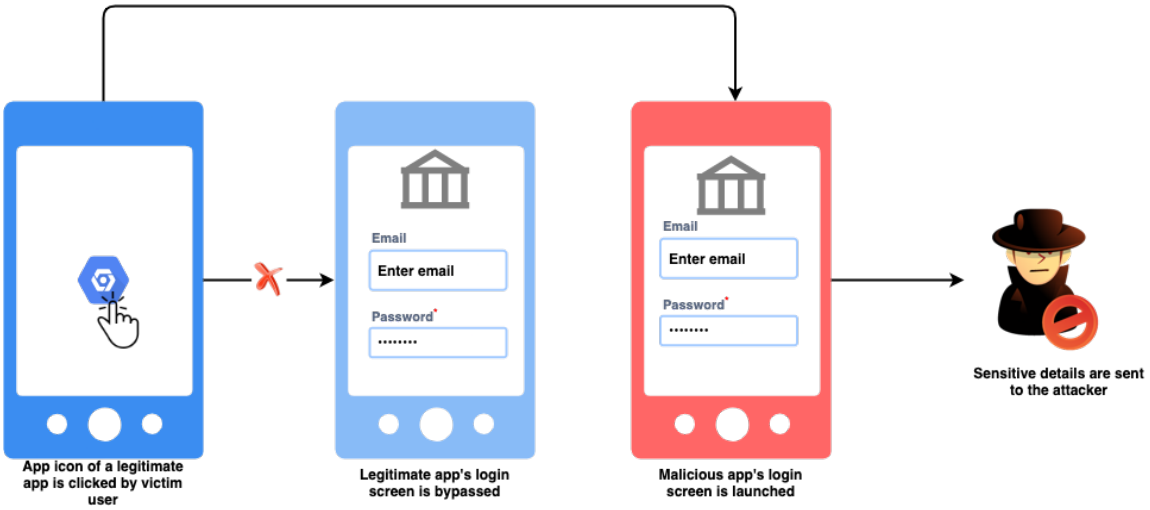

Figure 9: Strandhogg 2.0, a malicious app is started instead of legitimate app and attempts to capture user credential.

4.7 Cryptographic Vulnerabilities

Cryptographic algorithms provide security protection of contents when it falls into the wrong hand. However, vulnerabilities in using crypto APIs significantly reduce the protections in practice [30, 41]. Using both static and dynamic approaches, we obtain a comprehensive picture of crypto misuses by the wallet apps. We use Cryptoguard [70], a static analyzer that detects the existence of crypto API misuses in source code but can

not verify the vulnerable code that triggers at runtime. To overcome this issue, we use Crylogger [68], a dynamic analyzer that detects crypto misuses by instrumenting standard Java library classes. Both tools complement each other and expose a wide range of misuses by reducing the number of false positives. We observe 278/311 apps have at least one type of cryptographic misuses and 233/311 apps have at least two. For the summary of the Cryptoguard findings, see Table 6 and for the summary of Crylogger findings, see Table 7. No app in our data set has violated two Crylogger rules (Static seed for Pseudo-Random number generator (PRNG) and textbook algorithm of RSA), therefore we remove the two rules in the summary table.

To confirm that each cryptographic misuses identified by the tools apply to wallet apps, we manually verified the top 18 apps' decompiled code. We observe a significant number of issues are coming from 3rd party libraries used by the app, rather than from the app code itself. Some apps (e.g., asia.coins.mobile, com.paxful.wallet) use obfuscation techniques that hinder manual source code verification. The vulnerabilities identified and the impact of the vulnerabilities on wallet apps are as follows:

1. *Hardcoded Store Pass*: This violation indicates insecure coding styles where hard-coded secrets are stored inside the source code or in shared preferences. We find evidence of storing API key, API secret, and API session ID saved in preferences. In Android, an app can access only its keystore and it has been shown that privilege escalation attacks that circumvent this restriction allow exploitation of this vulnerability [86].

2. *Dummy Hostname Verifier*: Several libraries, such as, Apache HTTP, CleverTap [8] analytics do not verify hostname, which expands the possibility of the MITM attack surface. Apps like asia.coins.mobile, com.unocoin.unocoinwallet, org.toshi, and, zebpay.Application are depending on the mentioned libraries. There are several API client libraries available in the Android ecosystem (e.g. Apache HTTP, okhttp), and apps or 3rd-party libraries can use any API client libraries to communicate with a backend server via REST API. An instance of an API client library can be overridden to use a custom hostname verifier by the instance creator (e.g. apps or 3rd party libs) instead of its default hostname verifier. We observe the custom hostname verifier is misconfigured to accept any hostname and doesn't verify the requesting hostname and received certificate hostname. To verify the exploitability of this vulnerability, we create a custom certificate signed by mitmproxy and use the certificate to intercept network communication. We are able to capture the network communication initiated by the misconfigured API client instances.

3. *Used Improper Socket*: It detects if SSLSocket is directly used without performing hostname verification. In Java, a method named 'verify' (of class HostnameVerifier) is invoked to check if any hostname verification is incorporated. To avoid false-positive results, Cryptoguard does not consider the direct subclass implementation of SSLSocketFactory.

4. *Used HTTP*: Analytics libraries like Google Analytics, Appsflyer by Adobe, and CleverTap use HTTP connections to get static contents like images/icons and send

analytics events to their servers.

5. *Broken Symmetric Crypto Algorithm*: To encrypt key revealing information, some apps use broken crypto algorithm, such as 'AES/ECB/NoPadding', AES/CBC/PKCS 5Padding, Blowfish. In our manual investigation, we find a 3rd-party open source library bitcoinj[24] uses the 'AES/ECB/NoPadding' algorithm and some apps are using bitcoinj as the underlying library to handle cryptographic operations, which makes all such apps insecure. One app (com.mycelium.wallet) is using a TOR connection to connect with peer nodes and using a 3rd party library Orchid,[25] which uses 'AES/ECB/NoPadding' cipher to encrypt TOR stream. A popular Chinese library named Tencent is using broken RC4 to encrypt log files before sending them to the server. If the communication is intercepted, the app's usage pattern, activities, and operational data in log files can be exposed.

6. *Insecure Asymmetric Crypto*: This rule can detect insecure asymmetric cipher algorithm configurations, such as 1024-bit RSA algorithm, statically defined key, and insecure default key size. Cryptoguard cannot detect any instance of insecure asymmetric cipher in the top 18 apps.

7. *Broken Hash*: This vulnerability indicates the usage of insecure cryptographic hash functions (e.g., SHA1, MD5, MD4, MD2). In the Orchid library, SHA-1 is used for TOR message digest and com.mycelium.wallet app is using Orchid library as TOR client. Hash collisions of these algorithms allow attackers to break the integrity of

[24]https://github.com/bitcoinj/bitcoinj
[25]https://github.com/subgraph/Orchid

any intercepted messages [79]. Apps are using a broken hash algorithm to store user-provided PIN/password in the device. The hashed value of PIN/password is also used as the encryption key to encrypt key revealing information.

8. *Randomness of Keys*: This vulnerability covers static, or badly derived IV, static or the same salt usage multiple times, and the violation of RFC-8018 recommended 1000 iterations for Password Based Encryption (PBE) [62]. In our manual inspection, we find static or badly derived IV is a false positive result reported by the tools. Apps are using SecureRandom without any seed and therefore the seed is null. According to Android documentation both constructions of using SecureRandom, with and without seed, are valid usage. However, apps are using static or the same salt in the encryption of transaction information before sending it to the server.

4.8 Generic Vulnerabilities

We find several vulnerabilities, using multiple state-of-the-art static vulnerability analysis tools. These vulnerabilities are generally applicable to Android apps, not specifically to crypto wallet apps alone. We consider only critical marked issues to reduce false positives. For a summary of the result, see Table 8.

The vulnerabilities discovered and associated security risks in apps are the following:

1. *Implicit service*: Implicit intents do not specify a component name, rather declare an action. Any component, capable to act, can respond to the Intent [15]. For example, if the user taps on a link, any browser components installed in the device can respond

Application ID	*Predictable cryptographic keys*	*Predictable passwords for PBE*	*Hardcoded Store Pass*	*Dummy Hostname Verifier*	*Dummy Cert. Validation*	*Used Improper Socket*	*Used HTTP*	*Predictable Seeds*	*Untrusted PRNG*	*Static Salts*	*ECB mode for Symm. Crypto*	*Static IV*	*<1000 PBE iterations*	*Broken Symm. Crypto Algorithm*	*Insecure Asymm. Crypto*	*Broken Hash*
asia.coins.mobile		✗					✗					✗	✗			
co.bitx.android.wallet																
co.mona.android																
com.binance.dev																
com.bitcoin.mwallet	✗	✗		✗	✗	✗	✗				✗		✗	✗		
com.breadwallet																
com.coinomi.wallet	✗	✗	✗				✗		✗				✗			
com.mycelium.wallet	✗	✗	✗	✗	✗	✗	✗	✗		✗	✗	✗	✗	✗		✗
com.paxful.wallet	✗	✗	✗	✗		✗				✗			✗			
com.polehin.android		✗		✗		✗							✗			
com.unocoin.unocoinwallet	✗	✗	✗							✗			✗			
com.wallet.crypto.trustapp																
com.xapo																
de.schildbach.wallet	✗	✗		✗									✗			
exodusmovement.exodus	✗	✗	✗				✗		✗				✗			
org.toshi	✗	✗				✗	✗	✗			✗	✗	✗	✗		
piuk.blockchain.android																
zebpay.Application		✗	✗		✗			✗			✗		✗	✗		
Total	152	260	205	160	106	88	74	18	15	31	39	13	267	115	11	9

Table 6: Cryptographic vulnerabilities and misuses in wallet apps using Cryptoguard (static analysis). ✗ indicates the existence of the vulnerability in the app. Last row provides the total number for all 311 apps.

to act. Likewise, an implicit intent can trigger any service in the device that can perform declared action in the intent. Because, a service does not have a UI, so the user cannot be certain which service will respond to the intent and start. A malicious service that exists on the device can be activated and carry out undesirable actions.

2. *Exported ContentProvider*: Exported ContentProvider allows any other app on the device to access its controlled database. Critical information of the apps can be exposed if the data sharing logic is not implemented with caution, see Section 4.9.

Application ID	*Broken hash functions*	*Broken encryption alg.*	*Mode ECB with > 1 data block*	*Mode CBC (client/server scenarios)*	*Static key for encryption*	*Badly-derived key for encryption*	*Static initialization vector*	*Badly-derived initialization vector*	*Reuse initialization vector & key pairs*	*Static salt for key derivation*	*Short salt for key derivation*	*Same salt for different purposes*	*<1000 interations for key derivation*	*Weak password (score < 3)*	*NIST-black-listed password*	*Reuse password multiple times*	*Unsafe PRNG (java.util.Random)*	*Short key for RSA*	*Padding PKCS1-v1.5 for RSA*	*HTTP URL connections*	*Static password for store*	*Verify SSL hostnames trivially*	*Verify SSL certs trivially*	*Manually change hostname verifier*
asia.coins.mobile	✗			✗		✗											✗							
co.bitx.android.wallet	✗			✗	✗	✗	✗	✗	✗	✗		✗				✗	✗	✗						
co.mona.android	✗			✗	✗	✗	✗			✗		✗				✗	✗							
com.binance.dev																								
com.bitcoin.mwallet	✗																✗							
com.breadwallet	✗					✗		✗	✗								✗							
com.coinomi.wallet	✗	✗							✗	✗	✗	✗		✗		✗	✗							
com.mycelium.wallet	✗																✗						✗	
com.paxful.wallet	✗	✗			✗	✗											✗							
com.polehin.android	✗																✗							
com.unocoin.unocoinwallet	✗																✗							
com.wallet.crypto.trustapp																								
com.xapo	✗																✗							
de.schildbach.wallet	✗																✗							
exodusmovement.exodus	✗			✗				✗									✗							
org.toshi																								
piuk.blockchain.android	✗																✗							
zebpay.Application	✗		✗		✗			✗	✗	✗	✗	✗				✗	✗							
Total	232	6	11	56	31	55	18	43	27	19	3	13	1	4	1	19	162	23	4	7	2	1	13	5

Table 7: Dynamically detecting cryptographic misuses in wallet apps (using Crylogger). ✗ indicates the existence of violation of cryptographic rules in the app. Last row provides the total number for all 311 apps.

3. *URL not under SSL*: HTTP protocol is used to communicate with services.

4. *CVE-2013-4710* [6]: The target app has the method, addJavascriptInterface, which can be abused to allow JavaScript to control the host app.

5. *Runtime command execution*: The use of the critical function `Runtime.getRuntime().exec("...")`, which can be abused for command injection attack [3].

6. *CVE-2013-6271* [7]: The `isValidFragment` method in every PreferenceActivity class must be called to avoid exception throwing in Android 4.4.

7. *World read/writable file*: MODE_WORLD_READABLE or MODE_WORLD_WRITEABLE allows the file to be accessible by other applications. This vulnerability is identified as insecure data storage by OWASP [5].

8. *Insecure Pending Intent*: Intent should be explicit to specify an explicit component to be delivered, otherwise a malicious application could potentially intercept, redirect, and/or modify implicit Intent. Pending Intent is a wrapper over regular Intent to delegate a task to be performed by another app at a later time. Pending Intent contains the same authorization right and identity of the originating app. When the pending intent does not explicitly define a target component, then any app in the device, capable of handling the intent, becomes a potential responder. If a malicious app is selected by the user to handle the Intent then the malicious app will have all permissions and identity of the intent originating app. If there is only one app is the device capable to handle the intent then the app is selected by default without any interaction from the user's end.

9. *Insecure broadcast intent*: App that registers a broadcast receiver dynamically is vulnerable to granting unrestricted access to the broadcast receiver. The receiver will be called with any broadcast Intent that matches the filter.

10. *ECB cipher usage*: Electronic Code Book (ECB) mode does not provide good confidentiality, hence insecure, because, ECB mode produces the same output for the same input each time [4]. For example, when a user sends a password, the encrypted value is always the same. This makes it possible for an attacker to intercept and

replay the data.

11. *Encryption key in source*: Encryption keys are packaged with the application.

12. *RSA cipher usage*: RSA algorithm is used as the cipher mode without padding, which is insecure [1, 9].

Application ID	*Implicit service*	*Exported content provider*	*URL not under SSL*	*CVE-2013-4710*	*Runtime command exec.*	*CVE-2013-6271*	*World read/writeable file*	*Insecure pending intent*	*Insecure broadcast intent*	*ECB cipher usage*	*Encryption key in source*	*RSA cipher usage*
asia.coins.mobile	✗	✗	✗	✗				✗	✗			
co.bitx.android.wallet	✗		✗					✗	✗			
co.mona.android	✗							✗	✗	✗	✗	✗
com.binance.dev	✗	✗										
com.bitcoin.mwallet	✗			✗				✗	✗	✗	✗	
com.breadwallet					✗							
com.coinomi.wallet	✗		✗	✗	✗			✗	✗	✗	✗	
com.mycelium.wallet	✗		✗	✗				✗	✗	✗		
com.paxful.wallet			✗	✗	✗			✗	✗	✗		
com.polehin.android			✗	✗	✗	✗		✗	✗			
com.unocoin.unocoinwallet			✗	✗			✗	✗	✗	✗		
com.wallet.crypto.trustapp	✗		✗					✗	✗	✗		
com.xapo								✗	✗	✗	✗	✗
de.schildbach.wallet			✗					✗		✗	✗	
exodusmovement.exodus			✗	✗	✗			✗	✗	✗		
org.toshi	✗		✗	✗				✗	✗	✗	✗	
piuk.blockchain.android	✗							✗		✗		
zebpay.Application	✗	✗	✗	✗				✗				
Total	82	20	124	104	42	9	7	145	150	121	71	20

Table 8: Generic vulnerabilities in wallet apps. ✗ indicates the existence of the vulnerability in the app.

In theory, crypto algorithms may provide assurance to protect confidentiality (via encryption) and data integrity (via hashing); however, poor implementations of crypto algorithms can compromise the security of the communication. While dynamic crypto misuses detector tools, like Crylogger, cannot produce any false-positive result, because the data is collected from the API calls during runtime and it is evident that the API must be executed by the app's functionality. However, dynamic tools may produce false-negative results. On the other hand, statically detecting crypto misuses tools, like Cryptoguard, may produce a significant number of false-positive results. By using both static and dynamic approaches, we find a notable overlap in their misuses criteria and result-set; see Tables 6, 7, 8. For example, in Table 6, Cryptoguard identifies 152 app instances are using predictable cryptographic keys, whereas Crylogger found only 31 app instances for the same criterion. This observation indicates the shortcoming of static analysis tools, like Cryptoguard, infers any hardcoded key-like string as a potential encryption key, which may not be true and thus producing a false-positive result. Similarly, a dynamic analysis tool, such as Crylogger, cannot exhaustively navigate all the execution paths of an app and thus fails to identify all potential encryption keys. In another case, Crylogger identifies 162 apps are using unsafe PRNG (java.util.Random), whereas Cryptoguard identifies only 15 apps in this criterion. Due to code obfuscation, static analysis tools like Cryptoguard cannot identify the usage of a particular API, whereas dynamic tools can correctly identify the use of the API.

4.9 Exported Components

Depending on the responsibilities of a component, it may leak information or perform unauthorized tasks. Android Inter Process Communication (IPC) mechanism allows an app to share access with other apps on the device. If the IPC mechanism is not intended for use by other apps then, according to Android developer guidelines, the component must not be exported [16]. For example, Content providers offer a structured mechanism to allow other apps to access stored data. If a content provider is exported, it can reveal sensitive database information to any other app on the device. Similarly, if Activity and Service are not restricted then other apps can launch the activity or bind to the service, which may allow a malicious app to gain access to sensitive information, perform unauthorized actions, or corrupt the internal state of the app [2]. We find each one of the top wallet apps exports from 3 to 25 components, expanding the attack surface on wallet apps.

Application ID	*Activity*	*Service*	*Receiver*	*Provider*
asia.coins.mobile	2	1	5	1
co.bitx.android.wallet	5	2	2	0
co.mona.android	5	2	4	0
com.binance.dev	6	6	10	3
com.bitcoin.mwallet	3	3	2	0
com.breadwallet	1	1	2	0
com.coinomi.wallet	6	2	2	0
com.mycelium.wallet	2	6	1	1
com.paxful.wallet	2	1	2	0
com.polehin.android	2	1	3	0
com.unocoin.unocoinwallet	5	0	2	0
com.wallet.crypto.trustapp	2	3	3	1
com.xapo	4	3	2	0
de.schildbach.wallet	4	0	2	0
exodusmovement.exodus	2	1	3	0
org.toshi	3	2	4	0
piuk.blockchain.android	1	1	1	0
zebpay.Application	3	3	2	1

Table 9: Exported components summary for top 18 wallet apps. The values indicate the number of exported components of a particular type in the app.

Chapter 5

Evaluation

In this section, we discuss Horus's current capabilities, limitations, and the challenges we faced in our work. Horus's static module is based on the app call graph, enabling the framework to survive the code obfuscation. Android obfuscation techniques work on the app code and not on the SDK APIs. We note that Android API calls remain unobfuscated in the call graph. Our dynamic module is based on the app's artifacts and network trace. The dynamic module does not have any dependency on the app platform and source code. The artifacts analysis technique is equally effective in apps developed using hybrid and cross-platform technologies. Overall, Horus can quickly assess the security standards followed by a crypto wallet app using the static module, and provide a deeper understanding of the app's sensitive data handling process using the dynamic module.

5.1 Limitations and Challenges

Static analysis tools suffer from obvious limitations. They can only determine whether specific APIs or syntax patterns are present in the source code but cannot indicate whether the implementation is error-free. Depending on the syntax pattern, Horus may indicate that an app has implemented a security feature, but in reality, the implementation may be flawed and may still contain serious bugs. The dynamic analysis module of Horus does not take into account all encryption algorithms supported by the Android platform and thus not comprehensive. We also do not emulate a scenario when a salt value (static or dynamic) is used in generating an encryption key.

To make Horus fully automated, we evaluate tools like Monkey[26] and Droidbot[27] to perform sign-up, import wallet, and complete a transaction. However, in our evaluation, we find that the pseudo-random events that Monkey generates cannot accomplish a set of pre-defined tasks. It is possible to accomplish specific tasks using Droidbot, but it is not well-suited for a generalized workflow. We have to write a customized script for each app. Considering the apps' versatility, writing a script for each app beats the purpose of an automated framework, and we settled for a semi-automated solution.

We identify some wallet apps that encrypt key revealing information, use salt with the encryption key. We find hard-coded salt values in the source code and salt values printed in logs. In Horus we are not automating when salt is used in the encryption key.

[26] https://developer.android.com/studio/test/monkey
[27] https://github.com/honeynet/droidbot

Starting with Android 7.0 (API level 24), system-wide private certificates are not accepted in Android. Optionally, an app can explicitly choose to rely on a private certificate [31]. Also, as of Android 9.0 (API level 28), plaintext traffic is not allowed and cannot be configured as well [22]. To overcome this, we use mitmproxy[28] to monitor network traffic and place the mitmproxy certificate in the system certificate folder. Also, we use tcpdump to capture network requests in a pcap file to look for key revealing information in network communication.

5.2 Recommendations for Users

Based on our study, we make the following recommendations for the users of wallet apps.

- User should not use wallet apps in a rooted device; the device and OS's default protection is ineffective in a rooted device.

- Trusted app store is an effective protection mechanism for any category of apps. An app goes through numerous security checks before being released to the public [46]. However, there are many unofficial stores where an app can be downloaded. A repackaged app can pose a security hazard by emulating the real app behavior but internally capture the user's sensitive data. Users should be more security-aware about the risk of unofficial stores.

- Taking screenshots is a convenient feature but can introduce significant security risks, and sensitive data can be exposed if the user or any 3rd-party app takes a screenshot

[28] https://mitmproxy.org/

while sensitive data is displayed. Users should be aware of such actions' security risks.

- When the user types in a text field, the input is no secret to the keyboard app. The open nature of Android enables the user to use any 3rd-party app to replace the default keyboard. Even if the user uses the default keyboard, there are still security issues by malicious apps having virtual keyboard permission. Users should understand the security risk that comes with replacing the default keyboard option and providing input permissions to an untrusted app.

5.3 Recommendations for Developers

Based on our study, we make the following recommendations for the developers of wallet apps.

- A wallet app should detect a rooted device, and upon positive detection, the app should display a blocking user alert explaining the potential security risks of using a financial app in a rooted device and exit.

- Developers should implement an integrity checking mechanism in the app to verify the app's integrity before starting its operations.

- Taking screenshots should be disabled while the app is operating in the foreground.

- A custom secure keyboard should be implemented in the app to take the key revealing information from the user.

- App developers should choose the recommended secure random number generator available in the platform. For Android, which uses a modified version of Linux kernel, developers should use /dev/urandom [85] to generate seed for SecureRandom [23], a cryptographically strong random number generator.

- Developers have to be extremely cautious in implementing crypto APIs and follow API guidelines strictly. In addition to that, developers should be more security conscious and develop a perspective to see plausible security vulnerabilities in apps and not just direct attacks. In a survey [68], developers argue that sensitive data can be incorrectly encrypted if it is stored only locally because privilege escalation is required to access it. However, side-channel attacks occur frequently and can pose significant security threats to apps.

- 2FA provides an additional layer of security and protects the user from compromised credentials. Biometric authentication is a popular 2FA choice along with the PIN/password in the Android ecosystem. This additional security of 2FA should be implemented and utilized before performing any sensitive operations.

- Hardware security module is a separate computing environment, which acts as a safeguard of key revealing information. It is available on relatively newer and high-end devices and should be utilized (if available) to store the key revealing information on the device.

- Exported components should be minimized as much as possible, and this is especially important for financial apps like a wallet. Exported components are used to

communicate between apps in the device. Crypto wallet apps contain key revealing information and should act in isolation as much as possible to avoid any potential data disclosure risk.

Chapter 6

Conclusion and Future Work

With the massive growth of cryptocurrencies, the number of threat vectors on crypto wallet apps is also increasing. This puts millions of users at risk if security concerns are not adequately addressed in leading wallet apps. We introduced Horus, a semi-automated framework to analyze and detect security issues in crypto wallet apps. We analyzed 311 apps on the Android platform and discover a unique set of vulnerabilities. Our analysis indicates that security standards are not followed when developing apps, and there are vulnerabilities in the protection of key revealing information. Serious security gaps appear in popular wallet apps, including asking for dangerous permissions without a proper need. Based on our analysis, there is a lack of checks and balances in our understanding of wallet apps' security and their actual implementation. Users should be more vigilant and proactive in evaluating apps before relying on them. Additionally, developers should be better informed about the industry's security standards and strictly adhere to best practices and recommendations.

For future work, we can expand our framework to include iOS wallet apps into our analysis. We plan to incorporate all the encryption algorithms supported by the Android platform in Horus, such as ChaCha20, Digital Signature Algorithm (DSA), Triple Data Encryption Standard (DESede), etc. We also plan to consider all hardcoded strings found in the app codebase as potential encryption keys and potential salt values. Finally, we can include desktop wallet apps in our investigation and improve our design to support desktop apps.

Bibliography

[1] CWE-780: Use of RSA algorithm without OAEP (4.5), . URL https://cwe.mitre.org/data/definitions/780.html.

[2] CWE-926: Improper export of Android application components (4.5), . URL https://cwe.mitre.org/data/definitions/926.html.

[3] Command injection | OWASP. URL https://owasp.org/www-community/attacks/Command_Injection.

[4] Insecure use of cryptography | GuardRails. URL https://docs.guardrails.io/docs/en/vulnerabilities/java/insecure_use_of_crypto.

[5] M2: Insecure data storage | OWASP. URL https://owasp.org/www-project-mobile-top-10/2016-risks/m2-insecure-data-storage.

[6] Nvd - cve-2013-4710, . URL https://nvd.nist.gov/vuln/detail/CVE-2013-4710.

[7] Nvd - cve-2013-6271, . URL https://nvd.nist.gov/vuln/detail/CVE-2013-6271.

[8] Omnichannel customer engagement & user retention platform | CleverTap. URL https://clevertap.com/.

[9] Why RSA encryption padding is critical | rdist. URL https://rdist.root.org/2009/10/06/why-rsa-encryption-padding-is-critical/.

[10] NVD - CVE-2020-0096, 2020. URL https://nvd.nist.gov/vuln/detail/CVE-2020-0096.

[11] Testnet - Bitcoin Wiki, 2020. URL https://en.bitcoin.it/wiki/Testnet.

[12] Fabio Aiolli, Mauro Conti, Ankit Gangwal, and Mirko Polato. Mind your wallet's privacy: Identifying Bitcoin wallet apps and user's actions through network traffic analysis. In *34th ACM/SIGAPP Symposium on Applied Computing*, pages 1484–1491, 2019.

[13] Aisha I Ali-Gombe, Brendan Saltaformaggio, Dongyan Xu, and Golden G Richard III. Toward a more dependable hybrid analysis of Android malware using aspect-oriented programming. *Computers & Security*, pages 235–248, 2018.

[14] Emad Almutairi and Shiroq Al-Megren. Usability and security analysis of the Keepkey wallet. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 149–153. IEEE, 2019.

[15] Android Developers. Intents and Intent filters, . URL https://developer.android.com/guide/components/intents-filters#Types.

[16] Android Developers. Use interprocess communication, . URL https://developer.android.com/training/articles/security-tips#IPC.

[17] Android Developers. Shrink, obfuscate, and optimize your app, 2019. URL https://developer.android.com/studio/build/shrink-code#obfuscate.

[18] Android Developers. Back up user data with auto backup, 2021. URL https://developer.android.com/guide/topics/data/autobackup.

[19] Android Developers. HSM - Hardware Security Module, 2021. URL https://developer.android.com/training/articles/keystore.

[20] Android Developers. Create an input method, 2021. URL https://developer.android.com/guide/topics/text/creating-input-method.html.

[21] Android Developers. Dangerous permissions, 2021. URL https://developer.android.com/reference/android/Manifest.permission.

[22] Android Developers. Network security configuration, 2021. URL https://developer.android.com/training/articles/security-config.

[23] Android Developers. SecureRandom, 2021. URL https://developer.android.com/reference/java/security/SecureRandom.

[24] Andreas M. Antonopoulos. *Mastering Bitcoin*. O'Reilly Media, Inc., 03 2021. ISBN 9781491954386.

[25] Myrto Arapinis, Andriana Gkaniatsou, Dimitris Karakostas, and Aggelos Kiayias. A formal treatment of hardware wallets. In *International Conference on Financial Cryptography and Data Security*, pages 426–445. Springer, 2019.

[26] Paul Bankhead. Android developers blog: Improving discovery of apps and games on the Play Store, 2018. URL https://android-developers.googleblog.com/2018/06/improving-discovery-of-quality-apps-and.html.

[27] Francesco Bergadano, Milena Boetti, Fabio Cogno, Valerio Costamagna, Mario Leone, and Marco Evangelisti. A modular framework for mobile security analysis. *Information Security Journal*, pages 220–243, 2020.

[28] Bitcoin. Github - bitcoin/BIPS: Bitcoin Improvement Proposals, 2011. URL https://github.com/bitcoin/bips.

[29] Bitcoin. How Bitcoin transactions work | how does Bitcoin work? | get started with Bitcoin.com, 2017. URL https://www.bitcoin.com/get-started/how-bitcoin-transactions-work.

[30] Dan Boneh. Twenty years of attacks on the RSA cryptosystem. *Notices of the AMS*, pages 203–213, 1999.

[31] Chad Brubaker. Android developers blog: Changes to trusted certificate authorities in Android Nougat, 2016. URL https://android-developers.googleblog.com/2016/07/changes-to-trusted-certificate.html.

[32] Wai Kok Chan, Ji-Jian Chin, and Vik Tor Goh. Evolution of Bitcoin addresses from security perspectives. In *2020 15th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 1–6. IEEE, 2020.

[33] Qi Alfred Chen, Zhiyun Qian, and Z Morley Mao. Peeking into your app without

actually seeing it: UI state inference and novel Android attacks. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 1037–1052, 2014.

[34] Catalin Cimpanu. Cryptocore hacker group has stolen more than $200m from cryptocurrency exchanges | ZDNet, 2020. URL https://www.zdnet.com/article/cryptocore-hacker-group-has-stolen-more-than-200m-from-cryptocurrency-exchanges/.

[35] Catalin Cimpanu. Bitcoin wallet update trick has netted criminals more than $22 million | ZDNet, 2020. URL https://www.zdnet.com/article/bitcoin-wallet-trick-has-netted-criminals-more-than-22-million/.

[36] Wenrui Diao, Xiangyu Liu, Zhe Zhou, Kehuan Zhang, and Zhou Li. Mind-reading: Privacy attacks exploiting cross-app KeyEvent injections. In *European Symposium on Research in Computer Security*, pages 20–39. Springer, 2015.

[37] Wenrui Diao, Xiangyu Liu, Zhou Li, and Kehuan Zhang. No pardon for the interruption: New inference attacks on Android through interrupt timing analysis. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 414–432. IEEE, 2016.

[38] Alexander Druffel and Kris Heid. Davinci: Android app analysis beyond Frida via dynamic system call instrumentation. In *International Conference on Applied Cryptography and Network Security*, pages 473–489. Springer, 2020.

[39] Earlence Fernandes, Qi Alfred Chen, Justin Paupore, Georg Essl, J Alex Halderman,

Z Morley Mao, and Atul Prakash. Android UI deception revisited: Attacks and defenses. In *International Conference on Financial Cryptography and Data Security*, pages 41–59. Springer, 2016.

[40] Forensic Focus. Forensics and Bitcoin, 2015. URL https://www.forensicfocus.com/articles/forensics-bitcoin/.

[41] Christina Garman, Matthew Green, Gabriel Kaptchuk, Ian Miers, and Michael Rushanan. Dancing on the lip of the volcano: Chosen ciphertext attacks on Apple iMessage. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 655–672, 2016.

[42] Ioannis Gasparis, Zhiyun Qian, Chengyu Song, and Srikanth V Krishnamurthy. Detecting Android root exploits by learning from root providers. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1129–1144, 2017.

[43] Miraje Gentilal, Paulo Martins, and Leonel Sousa. Trustzone-backed Bitcoin wallet. In *Fourth Workshop on Cryptography and Security in Computing Systems*, pages 25–28, 2017.

[44] Clint Gibler, Jonathan Crussell, Jeremy Erickson, and Hao Chen. Androidleaks: Automatically detecting potential privacy leaks in Android applications on a large scale. In *International Conference on Trust and Trustworthy Computing*, pages 291–307. Springer, 2012.

[45] Andriana Gkaniatsou, Myrto Arapinis, and Aggelos Kiayias. Low-level attacks in

Bitcoin wallets. In *International Conference on Information Security*, pages 233–253. Springer, 2017.

[46] Google Developers. Play Protect, 2017. URL https://developers.google.com/android/play-protect/.

[47] Michael I Gordon, Deokhwan Kim, Jeff H Perkins, Limei Gilham, Nguyen Nguyen, and Martin C Rinard. Information flow analysis of Android applications in droidsafe. In *NDSS*, page 110, 2015.

[48] Mordechai Guri. Beatcoin: Leaking private keys from air-gapped cryptocurrency wallets. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1308–1316. IEEE, 2018.

[49] Gus Gutoski and Douglas Stebila. Hierarchical deterministic Bitcoin wallets that tolerate key leakage. In *International Conference on Financial Cryptography and Data Security*, pages 497–504. Springer, 2015.

[50] Trevor Haigh, Frank Breitinger, and Ibrahim Baggili. If I had a million cryptos: Cryptowallet application analysis and a trojan proof-of-concept. In *International Conference on Digital Forensics and Cyber Crime*, pages 45–65. Springer, 2018.

[51] Daojing He, Shihao Li, Cong Li, Sencun Zhu, Sammy Chan, Weidong Min, and

Nadra Guizani. Security analysis of cryptocurrency wallets in Android-based applications. *IEEE Network*, pages 114–119, 2020.

[52] Yiwen Hu, Sihan Wang, Guan-Hua Tu, Li Xiao, Tian Xie, Xinyu Lei, and Chi-Yu Li. Security threats from Bitcoin wallet smartphone applications: Vulnerabilities, attacks, and countermeasures. In *Eleventh ACM Conference on Data and Application Security and Privacy (CODASPY '21)*, pages 89–100, 2021.

[53] Daniel Kachakil. Discovering and exploiting a vulnerability in Android's personal dictionary (CVE-2018-9375) | IOActive, 2018. URL https://ioactive.com/discovering-and-exploiting-a-vulnerability-in-androids-personal-dictionary/.

[54] Puneet Kumar Kaushal, Amandeep Bagga, and Rajeev Sobti. Evolution of Bitcoin and security risk in Bitcoin wallets. In *2017 International Conference on Computer, Communications and Electronics (Comptelix)*, pages 172–177. IEEE, 2017.

[55] Taehun Kim, Hyeonmin Ha, Seoyoon Choi, Jaeyeon Jung, and Byung-Gon Chun. Breaking ad-hoc runtime integrity protection mechanisms in Android financial apps. In *2017 ACM on Asia Conference on Computer and Communications Security*, pages 179–192, 2017.

[56] Wiebe Koerhuis, Tahar Kechadi, and Nhien-An Le-Khac. Forensic analysis of privacy-oriented cryptocurrencies. *Forensic Science International: Digital Investigation*, page 200891, 2020.

[57] Cong Li, Daojing He, Shihao Li, Sencun Zhu, Sammy Chan, and Yao Cheng.

Android-based cryptocurrency wallets: Attacks and countermeasures. In *2020 IEEE International Conference on Blockchain (Blockchain)*, pages 9–16. IEEE, 2020.

[58] Pellaeon Lin. Tiktok vs Douyin: A security and privacy analysis - The Citizen Lab, 2021. URL https://citizenlab.ca/2021/03/tiktok-vs-douyin-security-privacy-analysis/.

[59] Zhen Ling, Melanie Borgeest, Chuta Sano, Jazmyn Fuller, Anthony Cuomo, Sirong Lin, Wei Yu, Xinwen Fu, and Wei Zhao. Privacy enhancing keyboard: Design, implementation, and usability testing. *Wireless Communications and Mobile Computing*, 2017:13–29, 2017.

[60] Boyarov Maksim. What is Wallet Import Format (WIF)?, 2019. URL https://allprivatekeys.com/what-is-wif.

[61] Antonio Marcedone, Rafael Pass, and Abhi Shelat. Minimizing trust in hardware wallets with two factor signatures. In *International Conference on Financial Cryptography and Data Security*, pages 407–425. Springer, 2019.

[62] Kathleen Moriarty, Burt Kaliski, and Andreas Rusch. PKCS# 5: Password-based cryptography specification version 2.1. *Internet Eng. Task Force (IETF)*, 8018:1–40, 2017.

[63] Will Neal. Cryptocurrency hackers steal $3.8 billion in 2020, 2021. URL https://www.occrp.org/en/daily/13627-cryptocurrency-hackers-steal-3-8-billion-in-2020.

[64] Emma Newburger. Bitcoin surpasses $60,000 in record high as rally accelerates,

2021. URL https://www.cnbc.com/2021/03/13/bitcoin-surpasses-60000-in-record-high-as-rally-accelerates-.html.

[65] Charlie Osborne. AT&T dragged to court, again, over SIM hijacking and cryptocurrency theft | ZDNet, 2020. URL https://www.zdnet.com/article/at-t-dragged-to-court-again-over-sim-hijacking-and-cryptocurrency-theft/.

[66] Marek Palatinus and Pavol Rusnak. Official specification of BIP-0044, March 2019. URL https://github.com/bitcoin/bips/blob/master/bip-0044.mediawiki.

[67] Marek Palatinus, Pavol Rusnak, Aaron Voisine, and Sean Bowe. Official specification of BIP-0039, February 2021. URL https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki.

[68] L. Piccolboni, G. Di Guglielmo, L. P. Carloni, and S. Sethumadhavan. Crylogger: Detecting crypto misuses dynamically. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1972–1989. IEEE Computer Society, May 2021.

[69] Benjamin Powers. This elusive malware has targeted crypto wallets for a year - CoinDesk, January 2021. URL https://www.coindesk.com/elusive-malware-electrorat-targets-crypto-wallets.

[70] Sazzadur Rahaman, Ya Xiao, Sharmin Afrose, Fahad Shaon, Ke Tian, Miles Frantz, Murat Kantarcioglu, and Danfeng Yao. Cryptoguard: High precision detection of cryptographic vulnerabilities in massive-sized Java projects. In *2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2455–2472, 2019.

[71] Mishaal Rahman. Developers are facing huge drop in new installs after Play Store algorithm changes, June 2018. URL https://www.xda-developers.com/developers-huge-drop-new-installs-play-store-algorithm-changes/.

[72] Venkatesh-Prasad Ranganath and Joydeep Mitra. Are free Android app security analysis tools effective in detecting known vulnerabilities? *Empirical Software Engineering*, pages 178–219, 2020.

[73] Joel Reardon, Álvaro Feal, Primal Wijesekera, Amit Elazari Bar On, Narseo Vallina-Rodriguez, and Serge Egelman. 50 ways to leak your data: An exploration of apps' circumvention of the Android permissions system. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 603–620, 2019.

[74] Jamie Redman. The $700 million wallet crack: Bitcoin's 7th largest address is under constant attack – Bitcoin News, 2020. URL https://news.bitcoin.com/the-700-million-wallet-crack-bitcoins-7th-largest-address-is-under-constant-attack/.

[75] Chuangang Ren, Yulong Zhang, Hui Xue, Tao Wei, and Peng Liu. Towards discovering and understanding task hijacking in Android. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 945–959, 2015.

[76] Ashish Rajendra Sai, Jim Buckley, and Andrew Le Gear. Privacy and security analysis of cryptocurrency mobile applications. In *2019 Fifth Conference on Mobile and Secure Services (MobiSecServ)*, pages 1–6. IEEE, 2019.

[77] Manuel San Pedro, Victor Servant, and Charles Guillemet. Side-channel assessment of open source hardware wallets. *IACR Cryptol. Eprint Arch.*, page 401, 2019.

[78] Kai Sedgwick. Bitcoin address formats – Wallets Bitcoin News, 2019. URL https://news.bitcoin.com/everything-you-should-know-about-bitcoin-address-formats/.

[79] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. The first collision for full SHA-1. In *Annual International Cryptology Conference*, pages 570–596. Springer, 2017.

[80] Saurabh Suratkar, Mahesh Shirole, and Sunil Bhirud. Cryptocurrency wallet: A review. In *2020 4th International Conference on Computer, Communication and Signal Processing (ICCCSP)*, pages 1–7. IEEE, 2020.

[81] The Financial Express. Bitcoin rally takes market cap of over 6,000 cryptocurrencies to whopping new high of $1.24 trillion, February 2021. URL https://www.financialexpress.com/market/bitcoin-rally-takes-market-cap-of-over-6000-cryptocurrencies-to-whopping-new-high-of-1-24-trillion/2189730/.

[82] Tyler Thomas, Mathew Piscitelli, Ilya Shavrov, and Ibrahim Baggili. Memory FORESHADOW: Memory FOREnSics of HArDware CryptOcurrency Wallets - a tool and visualization framework. *Forensic Science International: Digital Investigation*, page 301002, 2020.

[83] Md Shahab Uddin, Mohammad Mannan, and Amr Youssef. Horus: A security assessment framework for Android crypto wallets. In *17th International Conference on Security and Privacy in Communication Networks*. Springer, 2021.

[84] Roman Unuchek. Android: To root or not to root | Kaspersky official blog, 2017. URL https://www.kaspersky.com/blog/android-root-faq/17135/.

[85] urandom. /dev/random - Wikipedia, 2021. URL https://en.wikipedia.org/wiki//dev/random.

[86] Victor Van Der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clémentine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. Drammer: Deterministic rowhammer attacks on mobile platforms. In *2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1675–1689, 2016.

[87] Tejaswi Volety, Shalabh Saini, Thomas McGhin, Charles Zhechao Liu, and Kim-Kwang Raymond Choo. Cracking Bitcoin wallets: I want what you have in the wallets. *Future Generation Computer Systems*, pages 136–143, 2019.

[88] Artemij Voskobojnikov, Oliver Wiese, Masoud Mehrabi Koushki, Volker Roth, and Konstantin Beznosov. The U in crypto stands for usable: An empirical study of user experience with mobile cryptocurrency wallets. In *2021 CHI Conference on Human Factors in Computing Systems*, pages 1–14, 2021.

[89] Pieter Wuille. Official specification of BIP-0032, August 2020. URL https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki.

[90] Mingyuan Xia, Lu Gong, Yuanhao Lyu, Zhengwei Qi, and Xue Liu. Effective real-time Android application auditing. In *2015 IEEE Symposium on Security and Privacy*, pages 899–914. IEEE, 2015.

[91] Zhi Xu, Kun Bai, and Sencun Zhu. Taplogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors. In *Fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*, pages 113–124, 2012.

[92] Hang Zhang, Dongdong She, and Zhiyun Qian. Android root and its providers: A double-edged sword. In *22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1093–1104, 2015.

[93] Stephan Zollner, Kim-Kwang Raymond Choo, and Nhien-An Le-Khac. An automated live forensic and postmortem analysis tool for Bitcoin on Windows systems. *IEEE Access*, pages 158250–158263, 2019.