verichains

SECURITY AUDIT OF

RICE ANDROID WALLET

Rice Wallet

Public Report

Mar 17, 2022

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on Mar 17, 2022. We would like to thank Rice Wallet for trusting Verichains Lab in auditing the wallet extension. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the Rice Android Wallet. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified some vulnerable issues in the application, along with some recommendations.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY 5

1.1. About Rice Wallet 5

1.2. About Rice Android Wallet 5

1.3. Audit scope 5

1.4. Audit methodology 5

1.5. Disclaimer 6

2. AUDIT RESULT 7

2.1. Overview 7

2.2. Findings 7

2.3. Issues 8

2.3.1. PassCodeLockActivity.kt - Failed attempt locked time is bypassable CRITICAL 8

2.3.2. VerifyBackupWordActivity.kt - Wrong verify backup words CRITICAL 8

2.3.3. VerifyBackupWordActivity.kt - Verify backup words UI bug LOW 9

2.4. Possible enhancements 12

2.4.1. SettingDataRepository.kt - Zero bytes salt INFORMATIVE 12

2.4.2. WalletDataRepository.kt - Redundant code INFORMATIVE 12

2.4.3. Rooting INFORMATIVE 13

3. VERSION HISTORY 14

1. MANAGEMENT SUMMARY

1.1. About Rice Wallet

Rice Wallet is a non-custodial wallet that helps you store, invest and manage cryptocurrencies from one place with the best user experience.

Rice Wallet provides users with:

- An easy way to search and evaluate every single DeFi asset on the market.
- Buy & sell DeFi assets at the best prices from selected liquidity pools.
- Keep your portfolio in your pocket. Everything you need to manage your assets is available in a single app.

1.2. About Rice Android Wallet

Rice Android Wallet is the android version of Rice Wallet which was written in Kotlin Programming Language.

1.3. Audit scope

In this particular project, a timebox approach was used to define the consulting effort. This means that **Verichains Lab** allotted a prearranged amount of time to identify and document vulnerabilities. Because of this, there is no guarantee that the project has discovered all possible vulnerabilities and risks.

Furthermore, the security check is only an immediate evaluation of the situation at the time the check was performed. An evaluation of future security levels or possible future risks or vulnerabilities may not be derived from it.

This audit was conducted on commit 77d45df3a4958063e04be78316a742f4be2f6741 from git repository *https://github.com/rice-wallet/rice-android*.

1.4. Audit methodology

Verichains Lab's audit team mainly used the **Open Web Application Security Project (OWASP) Mobile Security Testing Guide (MTSG)**. The **MSTG** is a comprehensive manual for mobile app security development, testing and reverse engineering. It describes technical processes for verifying the controls listed in the **OWASP Mobile Application Verification Standard (MASVS)**. During the audit process, the audit team also used several tools for viewing, finding and verifying security issues of the app, such as following:

#	Name	Version
1	Mobile Security Framework (MobSF)	v3.5.0 beta
2	Frida tools	14.2.13
3	Android Studio	Bumblebee 2021.1.1
4	Visual Studio Code	1.64.2
5	Android Debug Bridge (adb)	1.0.41

Table 1. Tools used for audit

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the application functioning; creates a critical risk to the application; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the application with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the application with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 2. Severity levels

1.5. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure application. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

The Rice Android Wallet was written in Kotlin Programming Language with some help from java library like TrustWalletCore which provides low-level cryptographic wallet functionality. With the help of TrustWalletCore and android KeyStore, the Rice Android Wallet keeps users mnemonic seed and private key securely in their device with pin code protected.

The main features of the Rice Android Wallet are:

- Creating wallets for Ethereum-compatible chains like ETH, BSC and Polygon. Manage wallets, send and receive tokens.
- Swap tokens using 1inch Network.
- Search and evaluate every single DeFi asset on the market.
- Manage portfolio.

2.2. Findings

During the audit process, the audit team found some vulnerabilities in the given version of Rice Android Wallet.

#	Issue	Severity
1	Failed attempt locked time is bypassable	CRITICAL
2	Wrong verify backup words	CRITICAL
3	Verify backup words UI bug	LOW

Audit team also suggested some possible enhancements.

#	Issue	Severity
1	Zero bytes salt	INFORMATIVE
2	Redundant code	INFORMATIVE
3	Rooting	INFORMATIVE

2.3. Issues

2.3.1. PassCodeLockActivity.kt - Failed attempt locked time is bypassable CRITICAL

Rice Android Wallet implements a lock mechanism which lock passcode attempt when someone enters wrong passcode too many times (password guessing/bruteforce attack), and the locked time (60 seconds) is defined as in the snippet bellow:

```
410  currentTimePassed = (System.currentTimeMillis() - it.time) / 1000
```

The problem is System.currentTimeMillis() of Android is untrusted, attackers can modify system datetime to a future timestamp (after release time) to bypass the time restriction and continue to try another passcode. This approach does not require rooting and can be automated. After bruteforce pin code, the attackers can use it to steal mnemonic seed and private key.

RECOMMENDATION

Instead of System.currentTimeMillis() based time measuring, it is recommended to implement secure date time measuring using native module combination of time-synchronization from trusted source and local real-time clocks APIs like SystemClock.elapsedRealtime and SystemClock.elapsedRealtimeNanos. These return the elapsed time since the system was booted, including time when the device goes to deep sleep. This clock is guaranteed to be monotonic and continues to tick even when the CPU is in power saving mode, so is the recommended basis for general purpose interval timing. Also, the 60 seconds lock time is quite short, consider increase it.

UPDATES

- *Mar 17, 2022*: This issue has been acknowledged and fixed by the Rice Wallet team in commit fcc9b797c31a7e5982ad7d551524ede26de63659.

2.3.2. VerifyBackupWordActivity.kt - Wrong verify backup words CRITICAL

In compareList function, the for loop will return after the first loop, so the function only compares the first word, not all the words like expected.

```
171  private fun compareList(list1: List<Word>, list2: List<Word>): Boole…
     an {
172      for (i in list1.indices) {
173          return list1[i].position == list2[i].position && list1[i].co…
     ntent == list2[i].content
174      }
175      return false
176  }
```

RECOMMENDATION

Changing the code like below:

```
private fun compareList(list1: List<Word>, list2: List<Word>): Boolean {
  for (i in list1.indices) {
    if (list1[i].position != list2[i].position || list1[i].content != lis…
  t2[i].content) {
        return false;
    }
  }
  return true
}
```

UPDATES

- *Mar 17, 2022*: This feature has been removed from the wallet in commit fcc9b797c31a7e5982ad7d551524ede26de63659.

2.3.3. VerifyBackupWordActivity.kt - Verify backup words UI bug LOW

In verify backup words screen, if we touch a word fast, it will be duplicated like in the image below.

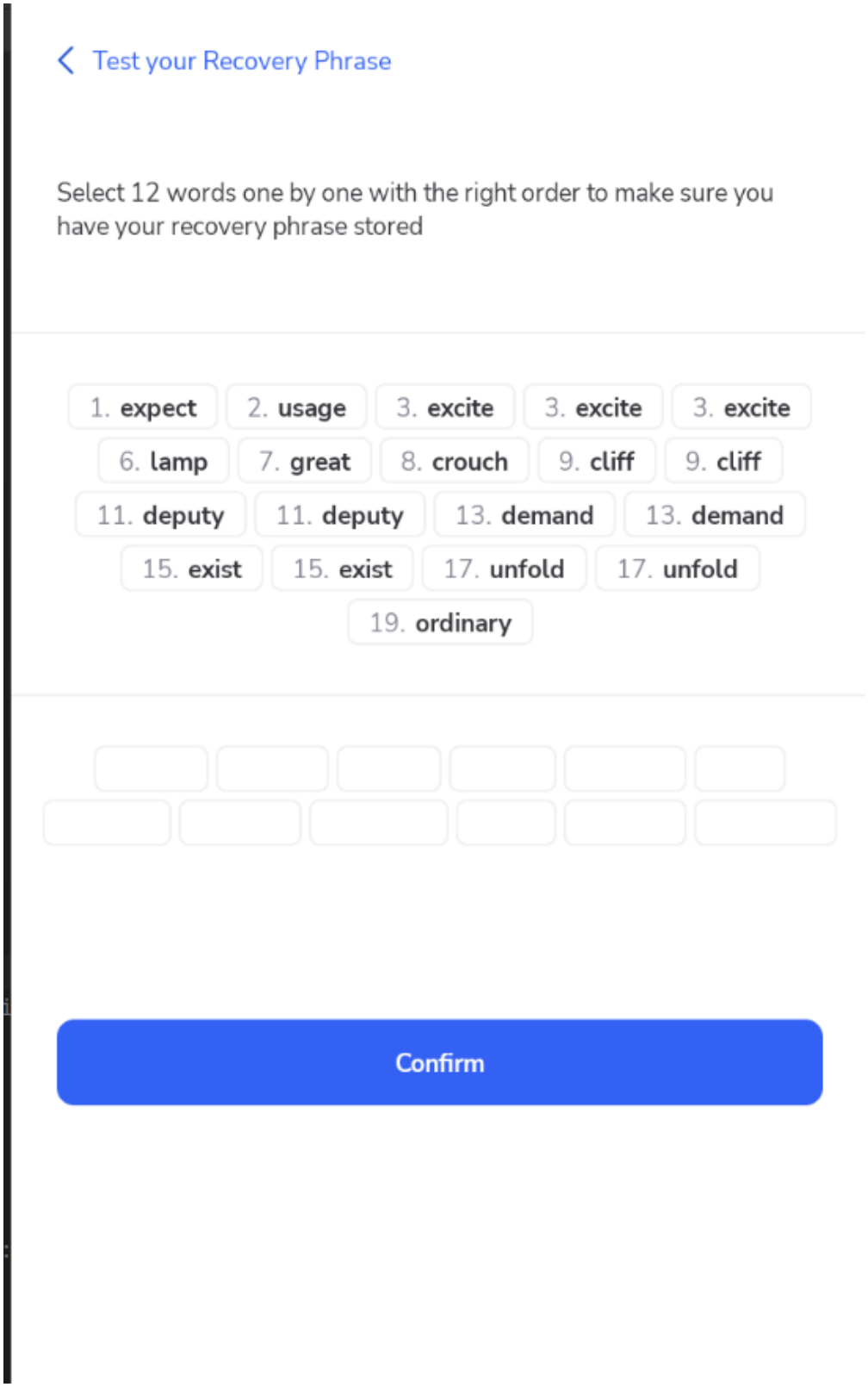
Test your Recovery Phrase
Select 12 words one by one with the right order to make sure you have your recovery phrase stored
1. expect
2. usage
3. excite
3. excite
3. excite
6. lamp
7. great
8. crouch
9. cliff
9. cliff
11. deputy
11. deputy
13. demand
13. demand
15. exist
15. exist
17. unfold
17. unfold
19. ordinary
Confirm

Image 1. Verify backup words UI bug

When wordAdapter1 is clicked, one verify backup word from currentWordList1 is removed and make it to be less than 12 words so no need to compareList. Instead, we should set btnNext.isEnabled to false if it is true. Also remove unused itemCount and position.

```
var itemCount = 0
var position = 0
wordAdapter2.setOnClick { selectedWord, pos ->
    currentWordList1.add(selectedWord.copy(isShow = true, type = 1))
    wordAdapter1.submitList(currentWordList1)
    wordAdapter1.notifyDataSetChanged()
    currentWordList2 = currentWordList2.mapIndexed { idx, value ->
        if (value.content == selectedWord.content)
            value.copy(isShow = false)
        else value
    }
    wordAdapter2.submitList(currentWordList2)
    wordAdapter2.notifyDataSetChanged()
    if (currentWordList1.size == 12) {
        if (compareList(currentWordList1, realList)){
            binding.btnNext.isEnabled = true
            binding.btnNext.alpha = 1f
        }
        Timber.e("checkCompare: ${compareList(currentWordList1, realList)…
  }")
    }
}

wordAdapter1.setOnClick { selectedWord, pos ->
    currentWordList2.find {
        it.position == selectedWord.position
    }?.isShow = true
    wordAdapter2.notifyDataSetChanged()

    currentWordList1.remove(selectedWord)
    wordAdapter1.notifyDataSetChanged()
    if (currentWordList1.size == 12) {
        if (compareList(currentWordList1, realList)){
            binding.btnNext.isEnabled = true
            binding.btnNext.alpha = 1f
        }
        Timber.e("checkCompare: ${compareList(currentWordList1, realList)…
```

```
  }")
    }
}
```

UPDATES

- *Mar 17, 2022*: This screen has been removed from the wallet in commit fcc9b797c31a7e5982ad7d551524ede26de63659.

2.4. Possible enhancements

2.4.1. SettingDataRepository.kt - Zero bytes salt INFORMATIVE

In hash function, there is redundant code when using zero bytes salt. If you don't want to add salt, just remove the salt update code.

```
68  private fun hash(passCode: String): String {
69      val salt = ByteArray(16)
70      val md = MessageDigest.getInstance("SHA-512")
71      md.update(salt)
72      return Numeric.toHexStringNoPrefix(md.digest(passCode.toByteArray…
    (StandardCharsets.UTF_8)))
73  }
```

RECOMMENDATION

Consider using a random salt or remove the zero bytes salt.

UPDATES

- *Mar 17, 2022*: This issue has been acknowledged by the Rice Wallet team.

2.4.2. WalletDataRepository.kt - Redundant code INFORMATIVE

There is redundant code in generatePassword function.

```
418  private fun generatePassword(): String {
419      val bytes = ByteArray(16)
420      val random = SecureRandom()
421      random.nextBytes(bytes)
422      Base64.encodeToString(bytes, Base64.DEFAULT)
423      return Base64.encodeToString(bytes, Base64.DEFAULT)
424  }
```

RECOMMENDATION

Removing redundant code.

UPDATES

- *Mar 17, 2022*: This issue has been acknowledged and fixed by the Rice Wallet team in commit fcc9b797c31a7e5982ad7d551524ede26de63659.

2.4.3. Rooting INFORMATIVE

On a rooted device, any applications with the root permission can read other app data. Rice Android Wallet uses Android room (which store in application data folder) to store hashed pin code. If users use the wallet on a rooted device, anyone got access to the device can read app data and get the hashed 6 number pin code (which they can easily bruteforce to recover the pin code) and then use the pin code to get users mnemonic seed and private key.

RECOMMENDATION

- Use KeyStore(Tink library) to securely encrypt/decrypt pin code.
- Prevent/warning users from running the wallet on rooted/emulation devices.

UPDATES

- *Mar 17, 2022*: Rice Wallet team has implemented a warning when running the wallet on rooted/emulation devices in commit fcc9b797c31a7e5982ad7d551524ede26de63659.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Feb 28, 2022*	Private Report	Verichains Lab
1.1	*Mar 17, 2022*	Public Report	Verichains Lab

Table 3. Report versions history