Digital Currency Wallet Application Security Audit Report

Loopr Wallet (Android)

SECBIT

Dec 15, 2018

1. Introduction

Loopr Wallet (Android) is a digital currency wallet application. SECBIT Labs conducted an audit from Nov 16th, 2018 to Dec 14th, 2018, including an analysis of the wallet (Android) in 3 areas: **common risks in Android applications**, **digital wallet assets security** and **server-side application security**. The assessment shows that Loopr Wallet (Android) has no critical security risks, and SECBIT Labs has some tips on logical implementation, potential risks and code revising (see Section 4 for details).

Type	Description	Level
Assets Security	mnemonic saved in clear text	Mid
Common App Risk	Insecure configuration in AndroidManifest.xml	Mid
Assets Security	Sensitive info like mnemonic recorded in log	Mid
Server-Side Risk	H5DexWebActivity uses http protocol for communication	Low
Common App Risk	No protection against sensitive information leak on inputs and user interface	Low
Common App Risk	No check for weak password and password retry	Low
Common App Risk	No encryption on stored files	Low
Common App Risk	No code obfuscation and consolidation for APK	Low
Common App Risk	No integrity check for APK	Low
Assets Security	No password protection for wallets imported by mnemonic without passwords	Low
Assets Security	Different private key generation processes on wallet creation between Android and iOS	Low
Assets Security	Variables storing private key are not deleted promptly	Info
Potential Risk	Not notify users that they cannot recover private keys from mnemonic after losing password	Info

2. Wallet Information

This section describes the basic wallet information.

Name	**Loopr Wallet**
System	Android
Released	No
Source	GitHub
File Type	Source Code, APK
Code Path	https://github.com/Loopring/loopr-android
Commit id	145bdb011e0f6037abd0828ae3a13e700c8b8dbc
Supported Tokens	ETH, ERC20 Token

3. Wallet Analysis

This section analyzes the functionality and security of the wallet application.

3.1 Functionality Analysis

As a cryptographic digital currency wallet, the implementation of Loopr Wallet (Android) in this area can be divided into 4 primary parts: wallet creation, wallet import, key management and asset transfer.

- Wallet Creation
    - Create one or multiple wallets.
    - Users can skip the mnemonic verification in the wallet creation.
- Wallet Import
    - Users can import wallets by mnemonic, Keystore and private keys.
- Key Management
    - Rename wallets.
    - Export Keystore files.
- Asset Transfer
    - Support sending and receiving ETH and ERC20 tokens.

3.2 Security Analysis

- Random Number Generation
    - Use SecureRandom as the secure source of randomness.
- Wallet Mnemonic Generation
    - Implementation satisfies BIP39. No issue has been found.
- Key Derivation
    - Use the open-source library BitcoinJ to derive keys. BitcoinJ has no known security issue, and follows BIP44 to derive keys with the correct coin type.
- Key Storage
    - Keys are encrypted by users' password and stored in the Keystore file. The open-source library Web3J is used to manage the Keystore file.
    - mnemonics are stored in clear text at some places, which is insecure.

- Key Management
  - Only allow exporting keys in the Keystore format and passwords are always required for the export.
- Input of Sensitive Info
  - No self-drawn keyboard for secure input.
  - Weak password in use.
- Secure Communication with Server
  - No private key are uploaded to the server.
  - Insecure http protocol in use.
- Common Android Security
  - Insecure AndroidManifest.xml configuration.
- Defend Screen Capture
  - No protection against the screen capture (partially fixed).

4. Audit Details

This section introduces the audit process and reports issues/risks/tips in detail.

4.1 Audit Process

The audit strictly follows the audit specification of SECBIT Labs. We analyze the project in 3 aspects: the common application security, the asset security and the server security. The audit is processed in following four steps:

- Each audit team reviews the wallet application independently.
- Each audit team evaluates the vulnerabilities and potential risks independently.
- Each audit team shares its audit results, and reviews results from other teams.
- All audit teams coordinate with the audit leader to prepare the final audit report.

4.2 Audit Result

After scanning by SECBIT internal tools and external open-source tools, the auditing team performed a manual assessment and inspected the source code of the wallet application. The audit results can be categorized into the following types:

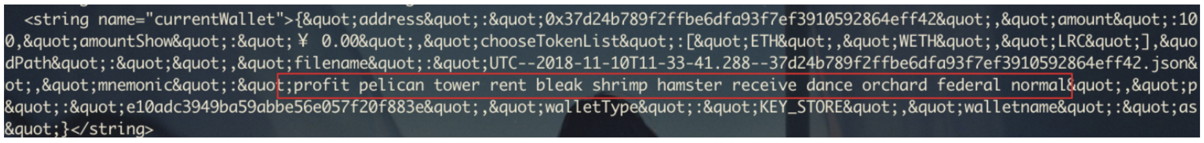
Number	Classification	Result
1	Check risks in wallet mnemonic generation and storage	Pass
2	Check risks in private key creation and storage	Pass
3	Check risks in local storage of critical info	Pass
4	Check risks in wallet import	Pass
5	Check risks in wallet password	Pass
6	Check risks in digital currency transfer	Pass
7	Check risks in private key and random number generation algorithm	Pass

8	Check risks in business logic	Pass
9	Check risks in user privileges	Pass
10	Check risks in App runtime environment	Pass
11	Check risks in App development procedure	Pass
12	Check risks in App components	Pass
13	Check risks in App local file and cache storage	Pass
14	Check risks in App and server networking	Pass

4.3 Issues

1. mnemonic saved in clear text

- Level: **Mid**
- Type: Assets Security
- Description:

  ```
   <string name="currentWallet">{&quot;address&quot;:&quot;0x37d24b789f2ffbe6dfa93f7ef3910592864eff42&quot;,&quot;amount&quot;:10
  0,&quot;amountShow&quot;:&quot;￥ 0.00&quot;,&quot;chooseTokenList&quot;:[&quot;ETH&quot;,&quot;WETH&quot;,&quot;LRC&quot;],&quo
  dPath&quot;:&quot;&quot;,&quot;filename&quot;:&quot;UTC--2018-11-10T11-33-41.288--37d24b789f2ffbe6dfa93f7ef3910592864eff42.json&
  ot;,&quot;mnemonic&quot;:&quot;profit pelican tower rent bleak shrimp hamster receive dance orchard federal normal&quot;,&quot;p
  &quot;:&quot;e10adc3949ba59abbe56e057f20f883e&quot;,&quot;walletType&quot;:&quot;KEY_STORE&quot;,&quot;walletname&quot;:&quot;as
  &quot;}</string>
  ```

  mnemonic for generating user private key are stored in `shared_pref/share_data.xml` without encryption. In addition, `leaf.prod.app/files/keystore/mnemonic.txt` caches the most recent mnemonic.

  If users' devices are lost, above files may leak users' private keys.

- Impact:

  Private keys could be leaked under certain circumstances.

- Suggestions:
  - Do not store mnemonic. In case it is necessary, store them after encryption.
  - Do not cache mnemonic.

  Developers could refer to the process of the Keystore file generation, and use the user password to encrypt mnemonic:

    - Use `PBKDF2-SHA-256` or `Scrypt` algorithm to get derived keys from user password.
    - Use AES by derived keys for mnemonic encryption and get the ciphertext.
    - Store the ciphertext and iv variables generated during AES computation.

      Meanwhile, developers could encrypt all contents of `SharedPreference` referring to issue `7. No encryption on stored files`.
- Result:
    - Deleted `mnemonic.txt` for mnemonic storage.
    - Encrypted mnemonic storage.

2. Insecure configuration in AndroidManifest.xml

- Level: **Mid**
- Type: Common App Risk
- Description:
    - `android:allowBackup=true` allows users to export all App data.
    - `android:exported=true` exposes a majority part of Activity.
- Impact:

  Ease the reverse-engineering of App.
- Suggestions:
    - Disallow App data exporting by setting `android:allowBackup=false`.
    - Setting `android:exported=false` for unnecessary Activity.
- Result:

  Set `android:exported=false` for unnecessary Activity.

3. Sensitive info like mnemonic recorded in log

- Level: **Mid**
- Type: Assets Security
- Description:

  Used `LyqbLogger.log()` to log sensitive information such as mnemonic and private keys. For example, line 378 of `GenerateWalletActivity.java` logs mnemonic.
- Impact:

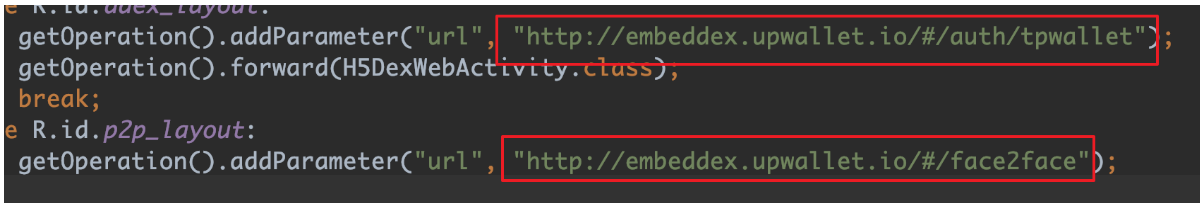
  Leak user private keys.

- Suggestions:

  Check and delete all code using `LyqbLogger.log()`.

4. H5DexWebActivity uses http protocol for communication

- Level: **Low**
- Type: Server-Side Risk
- Description:



```
e R.id.adex_layout:
 getOperation().addParameter("url", "http://embeddex.upwallet.io/#/auth/tpwallet");
 getOperation().forward(H5DexWebActivity.class);
 break;
e R.id.p2p_layout:
 getOperation().addParameter("url", "http://embeddex.upwallet.io/#/face2face");
```

  Communication via the http protocol is vulnerable to man-in-the-middle attacks.

- Impact:

  Attackers could forge H5 pages and steal user's assets and sensitive information.

- Suggestion:

  Use the https protocol for communication.

- Result:

  Applied https for key info communication.

5. No protection against sensitive information leak on inputs and user interface

- Level: **Low**
- Type: Common App Risk
- Description:

  No protection (such as anti-capturing, anti-recording or safe self-drawn keyboards) for inputing and displaying sensitive information (passwords and mnemonic).

- Impact:

  May leak user privacy info.

- Suggestions:
  - Anti-capturing/recording in the following interfaces:
    - mnemonic display during wallet creation and password entry
    - Private key and mnemonic export
    - Wallet import by private key or mnemonic and user password entry
    - User password entry during transferring, especially pop-ups

    - Use self-drawn keyboard instead of system keyboard.
- Result:

  Applied anti-capturing and anti-recording measures during wallet creation and exporting.

6. No check for weak password and password retry

- Level: **Low**
- Type: Common App Risk
- Description:

  No check and restriction for weak password and retry.
- Impact:

  User passwords might be vulnerable to the brute-force attack which can cause the assets stolen.
- Suggestions:
    - Notify users to enter strong passwords.
    - Restrict the number of password retries.
- Result:

  Notify users of weak passwords.

7. No encryption on stored files

- Level: **Low**
- Type: Common App Risk
- Description:

  Local user files in Android environment could be accessed without permission. Attackers will be able to access the sensitive user information in unencrypted files.
- Impact:

  May lead to the leak of sensitive user information and asset loss.
- Suggestions:

  Encrypt `SharedPreference` and local files (e.g. Keystore):
    - For `SharedPreference`, refer to secure-preferences https://github.com/scottyab/secure-preferences and encrypt keys and values of `SharedPreference`.
    - For local files, use AES implemented by the native code in C++:

        - Compared to Java, .so files generated from the native code are harder to be reverse-engineered and easier to consolidate.
        - Use functions to dynamically generate keys instead of hard-codeed in implementation of AES.
        - When calling encrypting and decrypting functions in .so files, dynamically get Android runtime environment and determine if the runtime environment is secure, which can increase difficulty of dynamic debugging.
        - Suggest users using strong wallet passwords, which can improve the security of the Keystore file and increase the difficulty to crack the Keystore file by brute-force.

8. No code obfuscation and consolidation for APK

- Level: **Low**
- Type: Common App Risk
- Description:

  No obfuscation and consolidation are performed for the APK, which will ease the reverse-engineering and analysis of App.
- Impact:

  Attackers could easily analyze App operations and explore hidden bugs for hacking.
- Suggestion:
    - Program `proguard-rules.pro` to obfuscate code when packing APK, especially code related to mnemonic and private keys.
    - Consolidate APK by related services before App releasing.

9. No integrity check for APK

- Level: **Low**
- Type: Common App Risk
- Description:

  APK does not verify the signature integrity, so it can be installed after re-signing.

  APK does not verify the file integrity, so it is still runnable after modification and re-signing.
- Impact:

  Attackers can replace the client App by a malicious one.

- Suggestion:
  - Perform integrity check for signature after running APK.
  - Apply common integrity check algorithms (CRC, MD5) to classes.dex and the whole APK.

10. No password protection for wallets imported by mnemonic without passwords

- Level: **Low**
- Type: Assets Security
- Description:

  Users could import password-less mnemonics (e.g. imToken App wallet). After importing, the wallet is not protected by passwords and others could make arbitrary transfers from the wallet. Also, passwords for Keystore files are null in such cases, which eases the stolen of private keys.
- Impact:

  May lead to asset loss.
- Suggestion:

  Provide additional password protection when importing mnemonic without passwords.
- Result:
  - Require mnemonic password as the wallet password when importing mnemonic.

11. Different private key generation processes on wallet creation between Android and iOS

- Level: **Info**
- Type: Assets Security
- Description:

  The iOS version of this wallet App uses an implementation of BIP39, which uses a password input by user for mnemonic, while the BIP39 implementation in the Android version does not use password for mnemonic. As a result, the two Apps can generate the same mnemonic with different private key seeds.
- Impact:

  The process to import wallet from Android to iOS and the process in the reverse direction become different.

- Suggestion:

  iOS and Android versions both use user password for mnemonic to generate private keys. Android will use the user password for mnemonic when generating private key seeds.

- Result:

  - Android version uses mnemonic passwords for wallet. Test shows that the wallet addresses generated by same mnemonic and passwords are the same now.

12. Variables storing private key are not deleted promptly

- Level: **Info**
- Type: Assets Security
- Description:

  Private keys are stored in String objects or Credential::BigInteger objects. When those objects are out of lives, they are not always immediately cleared from memory by GC. As a result, attackers may get users' private keys by dumping the memory.

- Impact:

  Private keys can be leaked.

- Suggestion:

  Use `[]bytes` to store private keys in memory and clear its contents after use promptly. A sample code snippet can be found below:

  ```
  bytes[] privKey = genPrivateKey();
  for (i:=0;i < privKey.length; i++){
      privKey[i] = 0;
  }
  ```

4.4 Risks

Risks are potential security issues in the use of App and the design logic of the product. The following risk is found by SECBIT Labs.

- Not notify users that they cannot recover private keys from mnemonic after losing password

- Level: **Info**
- Type: Potential Risk
- Description:

  Loopr Wallet (Android / iOS) uses mnemonic passwords, and users cannot recover private keys according to mnemonic without passwords. Common wallet apps support private key recovering by mnemonic and notify users to remember mnemonics. Those users may misunderstand Loopr wallet and forget to note their mnemonic passwords.
- Suggestion:

  Notify users by pop-ups during wallet creation that the user input password will be used as the mnemonic password and will be needed along with the mnemonic when recovering the wallet. The pop-ups should be confirmed and closed explicitly by users.

5. Conclusion

Loopr Wallet (Android) implements common wallet functions(account creating, key management, transfer) according to BIP32, BIP39 and BIP44 with additional functions by specific project targets. SECBIT Labs had found no critical bug or flaw after analyzing Loopr Wallet (Android). The wallet reveals some issues and potential risks in assets security, phone and server-side security as demonstrated above.

Disclaimer

Appendix

Vulnerability/Risk Level Classification

Level	Description
High	Severely damage the digital assets and allow attackers to steal, and/or disable digital assets of users.
Mid	Damage digital assets' security under limited conditions and cause impairment of benefit for stakeholders.
Low	Cause no actual impairment to digital assets.
Info	Relevant to security/best practice or rationality of digital assets and the wallet, not directly imply risks.