Fine penetration tests for fine websites

**Dr.-Ing. Mario Heiderich, Cure53**
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

# Pentest-Report DeBank Rabby iOS & Android Wallet 09.2024

Cure53, Dr.-Ing. M. Heiderich, BSc. C. Mayr

## Index

**Dr.-Ing. Mario Heiderich, Cure53**
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

## Introduction

*"Rabby Wallet- The game-changing wallet for Ethereum and all EVM chains"*

From https://debank.com/official/Rabby_Wallet

This report presents the results of a penetration test and source code audit against the DeBank iOS and Android Rabby Wallet mobile applications, as well as various wallet integrations. In context, the security-focused assessments were completed by Cure53 in CW38 September following the project request from DeBank representatives in August 2024. A total of ten days were invested to reach the coverage expected for this project.

Three separate Work Packages (WPs) were created to assemble manageable work proportions. These read as follows:

- **WP1**: White-box pen.-tests & code audits against Rabby Wallet iOS builds
- **WP2**: White-box pen.-tests & code audits against Rabby Wallet Android builds
- **WP3**: White-box pen.-tests & code audits against Rabby Wallet integrations

Cure53 was provided with mobile builds for both iOS and Android, all relevant sources for both applications, as well as other assorted data. The selected methodology was white-box and two experienced pentesters fulfilled the preparation, execution, and finalization stages of the assignment. Preliminary initiatives were actioned in advance (namely CW37 2024) to encourage a seamless start.

Communications were handled using Slack, with a dedicated channel established to combine the personnel from DeBank and Cure53. Cross-team discussions were smooth and minimal questions were required, since the scope was well prepared and clear. No noteworthy roadblocks were encountered at any point, contributing to an efficient and thorough examination period. Live reporting was not required, though the audit team still relayed a number of status updates about the general progress and related observations.

Regarding the findings, the Cure53 team detected fifteen after achieving extensive coverage over the elements in focus. Of those, six were assigned to the *Identified Vulnerabilities* section, while the other nine represented *Miscellaneous Issues* with lower exploitation potential.

Initially, the DeBank maintainers stated that their key area of concern pertained to the storage management of user private keys and mnemonics. Cure53 thus sought to ensure that all keys remain encrypted and stored in the Rabby Wallet, will not be uploaded by any network request, and cannot be obtained via remote or other malicious apps.

**Dr.-Ing. Mario Heiderich, Cure53**
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Consequently, the team focused on key handling and surrounding functionalities that may introduce vectors for leakage or retrieval. Cure53's efforts verified that DeBank has invested considerable resources into encrypting the passphrase and mnemonics at rest. However, during execution, both the passphrase and mnemonics are passed in plaintext through the application context.

Consequently, multiple findings were documented in which the mnemonic or passphrase could be retrieved via process dumps, as highlighted in RBY-01-001 and RBY-01-002. In addition, the RabbitCode secret is recoverable directly via the installer files, as described in ticket RBY-01-012. These findings were considered of *High* severity.

To effectively protect sensitive secrets from privileged attackers, robust safeguards must be implemented for handling this data during execution, which would additionally enhance the Wallet's overall security posture.

All in all, Cure53's estimation of the targets is mixed. While the apps are generally resilient, the excessive number of vulnerability ramifications suggests opportunities for security growth.

Onward, the report is divided into a number of key chapters for ease of reference. Firstly, the *Scope* provides all general setup information in concise bullet points. The document then lists all security problems in chronological order of detection (rather than degree of severity), grouped into two subcategories: *Identified Vulnerabilities* and *Miscellaneous Issues*. Each corresponding ticket offers a technical explanation, Proof-of-Concept (PoC) and/or steps to reproduce, code snippets, and remedial advice. Lastly, the *Conclusions* section summarizes Cure53's overarching viewpoints of the core focus facets, as well as substantiates their security effectiveness.

**Dr.-Ing. Mario Heiderich, Cure53**
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

## Scope

- **Pen.-tests & code audits against DeBank Rabby Wallet for iOS & Android**
  - **WP1:** White-box pen.-tests & code audits against Rabby Wallet iOS builds
    - **Source Code:**
      - https://github.com/RabbyHub/rabby-mobile/commit/a8dea5d8c530cb1acf9104a7854089256c36d85a
    - **Branch:**
      - audit/202409
    - **Mobile Application Builds:**
      - **Lark Link:**
        - https://debankglobal.larksuite.com/docx/S5UvdPWEZobdFDxQdcnuYxrRshh?from=from_copylink
  - **WP2:** White-box pen.-tests & code audits against Rabby Wallet Android builds
    - **Source Code:**
      - https://github.com/RabbyHub/rabby-mobile/commit/a8dea5d8c530cb1acf9104a7854089256c36d85a
    - **Branch:**
      - audit/202409
    - **Mobile Application Builds:**
      - **Lark Link:**
        - https://debankglobal.larksuite.com/docx/S5UvdPWEZobdFDxQdcnuYxrRshh?from=from_copylink
  - **WP3:** White-box pen.-tests & code audits against Rabby Wallet integrations
    - In scope are the various integrations that the mobile apps utilize, ranging from KYC to third party.
  - **Additional documentation:**
    - https://debankglobal.larksuite.com/docx/VjENdgDCaolxjZx6En7uBzrns0l
  - **Test-supporting material was shared with Cure53**
  - **All relevant sources were shared with Cure53**

**Dr.-Ing. Mario Heiderich, Cure53**
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

## Identified Vulnerabilities

The following section lists all vulnerabilities and implementation issues identified during the testing period. Notably, findings are cited in chronological order rather than by degree of impact, with the severity rank offered in brackets following the title heading for each vulnerability. Furthermore, all tickets are given a unique identifier (e.g., *RBY-01-001*) to facilitate any future follow-up correspondence.

### RBY-01-001 WP1-WP2: Mnemonic recoverable via process dump *(High)*

The audit team acknowledged that a user's mnemonic is recoverable from the DeBank Rabby iOS and Android mobile application's process memory, even after the user has actively closed the app and the device has been locked. The root cause of this flaw is likely related to the garbage collection mechanism, which retains data in memory until it is reclaimed.

Therefore, a malicious app with elevated privileges or a local attacker with *root* access on a victim's device could leverage this shortcoming to obtain the mnemonic. With this, one can successfully compute the private key of the wallet and take over the account, assuming control over any funds associated with the private key.

**Steps to reproduce:**

1. Apply the fridump[1] utility to dump the DeBank Rabby Wallet application's address space, as follows:

   ```
   $ python3 fridump.py -U -s "Reg Rabby Wallet"
   ```

2. Search for the first two words of the mnemonic following the dump creation.
3. Run *strings*[2] on the listed file to locate the mnemonic's clear-text storage, which is obtainable by dumping the memory while the app is locked:

   **Android:**
   ```
   [...]{"type":"HD Key Tree","data":{"mnemonic":"gossip loop runway
   lion cruise retreat rich liquid velvet bubble rural dwarf",[...]
   ```

   **iOS:**
   ```
   [...]
   steel tornado art gentle kit arrow cushion avoid omit garlic often
   autumn
   [...]
   ```

[1] https://github.com/Nightbringer21/fridump
[2] https://man7.org/linux/man-pages/man1/strings.1.html

**Dr.-Ing. Mario Heiderich, Cure53**
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

To mitigate this issue, Cure53 suggests protecting sensitive data (such as the user's mnemonic) in memory. This is considered fundamental in order to eliminate the plausible risk of malicious actors instigating a wallet takeover.

## RBY-01-002 WP1-WP2: Password recoverable via process dump *(High)*

The audit team confirmed that a user's password is recoverable from the DeBank Rabby iOS and Android mobile application's process memory, even after the user has actively closed the app and the device has been locked. Similarly to the previous ticket, this situation is primarily attributable to the garbage collection mechanism, which retains data in memory until it is reclaimed.

Therefore, a malicious app with elevated privileges or a local attacker with *root* access on a victim's device could leverage this shortcoming to obtain the password. With this, one can successfully unlock the app and obtain access to the private key.

**Steps to reproduce:**

1. Apply the fridump[3] utility to dump the DeBank Rabby Wallet application's address space, as follows:
   `$ python3 fridump.py -U -s "Reg Rabby Wallet"`

2. Search for the password following the dump creation.
3. Run *strings*[4] on the listed file to locate the password's clear-text storage, which is obtainable by dumping the memory while the app is locked:

   **Android:**
   ```
   aes-256-cbc
   rootsys1
   YjA5MjY3MDMwZDRmOGVmNWEwNDE2ZGZjMWY1ODNjNWM=
   [...]
   ```

   **iOS:**
   ```
   1.0344827586206897
   UUUUUU
   rootsys1
   [...]
   ```

To mitigate this issue, Cure53 suggests following the guidance offered in ticket RBY-01-001, which will help to avoid user account takeovers.

[3] https://github.com/Nightbringer21/fridump
[4] https://man7.org/linux/man-pages/man1/strings.1.html

**Dr.-Ing. Mario Heiderich, Cure53**
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

### RBY-01-010 WP1-WP2: Wallet auto lock bypass via system time change *(Medium)*

***Fix note:*** *This issue has been addressed by the development team and the fix has been verified by Cure53. The described problem no longer exists.*

During the source code audit, Cure53 discovered that the auto-lock timer relies on the local system time of the device to determine when to lock the wallet. Since this time can be manipulated by local attackers or malicious privileged apps running on the same device, an attacker could exploit this vulnerability to keep the wallet unlocked indefinitely, maintaining persistent access.

**Steps to reproduce:**

1. Open the DeBank Rabby Wallet application on either iOS or Android and complete the initial installation.
2. Navigate to the *Settings* window and observe that the default auto lock timer is set to 24 hours. At the bottom of the settings window, note that the counter *Last Unlock Offset* value increases as time progresses.
3. After 5 minutes, navigate to the system settings of the device and adjust the local time to 5 minutes before.
4. Revert back to the *Settings* window and observe that *Last Unlock Offset* has been reset, even though the app was not locked and unlocked again.

To mitigate this vulnerability, Cure53 recommends integrating the Android *ACTION_TIME_CHANGED* event[5] for Android. For iOS, a similar event likely exists and should be utilized. If this event is detected, the app could be locked regardless of the time.

### RBY-01-012 WP1-WP2: RabbitCode secret recoverable from installer files *(High)*

While conducting static and dynamic testing for the DeBank Rabby Wallet applications for iOS and Android, it was observed that both apps use a value referred to as *rabbitCode*. Since the purpose of this variable was not immediately obvious, Cure53 requested clarification from DeBank, who stated that:

The *rabbitCode* serves as a salt value for react-native-keychain, which is responsible for encrypting and decrypting sensitive information (such as the user-defined app password) from the OS-level keychain module. This encryption and decryption process must remain functional across multiple versions of Rabby Mobile to ensure that the data remains accessible after app updates, necessitating a static salt. Given its critical nature, the salt cannot be stored in any form (whether in plaintext or symmetrically encrypted), as both options pose security risks. Instead, the salt is embedded within the app's binary and obfuscated during the build process to prevent direct access. If the *rabbitCode* were to be leaked, it would pose a significant security threat.

[5] https://developer.android.com/reference/android/content/Intent

**Dr.-Ing. Mario Heiderich, Cure53**
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Given that this value is statically set during the build process, an assessment was performed to determine whether it could be easily retrieved from the installer packages. During this investigation, it was confirmed that a malicious actor can easily extract this sensitive value, as outlined below.

**Steps to reproduce:**

**For iOS:**

1. Install the DeBank Rabby Wallet application on a jailbroken iOS device.
2. Dump the decrypted IPA file using a tool such as TrollStore[6].
3. Download the decrypted IPA file from the device and unzip it.
4. Open the *Info.plist* file with an editor of choice and observe the static *rabbit_code* value, as shown below:

   **Content:**
   ```
   <key>rabbit_code</key>
   <string>X9[...]</string>
   ```

5. This value is also within the *Payload/RabbyMobile.app/AppConfig.xcconfig* file:

   **Content:**
   ```
   RABBY_MOBILE_CODE = X9[…]
   ```

**For Android:**

1. Open the APK installer file with a decompiler of choice, such as *jadx*[7].
2. Navigate to the MainActivity class inside the decompiled sources and observe the value stored in clear-text, unobfuscated:

   **Affected code:**
   ```
   public class MainActivity extends ReactActivity {

   /* loaded from: classes.dex */
   public class a extends j {
   public a(ReactActivity reactActivity) {
   super(reactActivity, "RabbyMobile");
   }

   @Override // o6.j
   public final Bundle a() {
   Bundle bundle = new Bundle();
   bundle.putString("rabbitCode", "X9[...]");
   return bundle;
   }
   ```

[6] https://trollstore.app/
[7] https://github.com/skylot/jadx

**Dr.-Ing. Mario Heiderich, Cure53**
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

```
}

[...]
}
```

As this value is deemed highly sensitive in nature, Cure53 recommends reconsidering the current design and adopting an alternative solution for storing the secret value within the Rabby Wallet application.

## RBY-01-013 WP2: Potential phishing via task hijacking on Android *(Low)*

***Fix note:*** *This issue has been addressed by the development team and the fix has been verified by Cure53. The described problem no longer exists.*

Testing confirmed that the Android app does not offer sufficient protection against task hijacking attacks.

The *launchMode* for the *MainActivity* activity is currently set to *singleTask*. While this mitigates task hijacking through StrandHogg 2.0[8] for Android API level 29 and below, the app remains vulnerable to older methods, such as the original StrandHogg[9] attack and other techniques documented since 2015[10].

The described vulnerability was patched by Google in March 2019 for Android versions 28 and newer. Since the Android app supports devices from Android 6 (API level 23), this renders all users running Android 6-8.1 vulnerable, as well as affects users running unpatched Android devices. The latter is still common in the modern era.

A malicious app could leverage this weakness to manipulate the way that users interact with the app. Specifically, this could be instigated by relocating a malicious attacker-controlled activity within the screen flow of the user, which may be useful toward performing phishing or Denial-of-Service (DoS) attacks.

**Affected file:**
*AndroidManifest.xml*

**Affected code:**

```
<activity android:label="@string/app_name"
android:name="com.debank.rabbymobile.MainActivity" android:exported="true"
android:launchMode="singleTask" android:screenOrientation="portrait"
android:configChanges="keyboard|keyboardHidden|orientation|screenLayout|
screenSize|smallestScreenSize|uiMode"
android:windowSoftInputMode="adjustPan">
```

[8] https://www.helpnetsecurity.com/2020/05/28/cve-2020-0096/
[9] https://www.helpnetsecurity.com/2019/12/03/strandhogg-vulnerability/
[10] https://s2.ist.psu.edu/paper/usenix15-final-ren.pdf

**Dr.-Ing. Mario Heiderich, Cure53**
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

```
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

To mitigate this issue, Cure53 advises implementing appropriate countermeasures. One solution would be to set the task affinity of the exported activity to an empty string via *android:taskAffinity=""*. This forces Android to create a random *name*, which any future attacker would find difficult to predict. Additionally, the *launchMode* can be set to *singleInstance*, which enforces the creation of a new task for each activity.

## RBY-01-014 WP1-WP2: Backup password prompt bypassable *(Medium)*

During the source code auditing phase, the ability to export the private key and/or mnemonic was analyzed. Users can export either of these values by selecting the corresponding wallet and then choosing either *Backup Seed Phrase* or *Backup Private Key* in the wallet/address details window. The user interface enforces a password re-entry before allowing the export of sensitive information. However, this password check can be bypassed by a privileged malicious app by hooking into the *throwErrorIfInvalidPwd* function and forcing it to return immediately, for instance. As a result, an error will not be thrown and both the *getMnemonics* and *getPrivateKey* functions will return the seed phrase and/or private key to the caller.

This attack can only be carried out by a privileged malicious application or a local attacker that has gained access to an unlocked device with the DeBank Rabby Wallet installed.

**Affected file:**
*rabby-mobile-a8dea5d8c530cb1acf9104a7854089256c36d85a/apps/mobile/src/core/apis/mnemonic.ts*

**Affected code:**

```
export const getMnemonics = async (password: string, address: string) => {
  await throwErrorIfInvalidPwd(password);
  const keyring = await keyringService.getKeyringForAccount(
    address,
    KEYRING_CLASS.MNEMONIC,
  );
  const serialized = await keyring.serialize();
  const seedWords = serialized.mnemonic;

  return seedWords;
};
```

**Dr.-Ing. Mario Heiderich, Cure53**
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

**Affected file:**
*rabby-mobile-a8dea5d8c530cb1acf9104a7854089256c36d85a/apps/mobile/src/core/apis/privateKey.ts*

**Affected code:**

```
export const getPrivateKey = async (
  password: string,
  { address, type }: { address: string; type: string },
) => {
  await throwErrorIfInvalidPwd(password);
  const keyring = await keyringService.getKeyringForAccount(address, type);
  if (!keyring) {
    return null;
  }
  return await keyring.exportAccount(address);
};
```

To mitigate this vulnerability, Cure53 advises altering the current logic to ensure that neither the private key nor seed phrase can be extracted by a privileged attacker, such as a malicious privileged app, without knowledge of the actual password used to decrypt these values.

**Dr.-Ing. Mario Heiderich, Cure53**
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

## Miscellaneous Issues

This section covers any and all noteworthy findings that did not incur an exploit but may assist an attacker in successfully achieving malicious objectives in the future. Most of these results are vulnerable code snippets that did not provide an easy method by which to be called. Conclusively, while a vulnerability is present, an exploit may not always be possible.

### RBY-01-003 WP1-WP2: Lack of rate limiting for password unlock *(Medium)*

Dynamic testing of the iOS and Android DeBank Rabby Wallet applications revealed that they lack adequate protection against brute-force attacks targeting the password used to unlock the app (and consequently the user's wallet). Thus, a user's password could potentially be brute forced. This problem owes to the fact that the application does not fully utilize *System Screen Lock* from the mobile platform (iOS and Android), which would effectively leverage SEP (iOS) or ARM TrustZone (Android) for verifying credentials such as the PIN, password, and biometrics.

This circumstance requires a local attacker with access to a device upon which the DeBank Rabby application has been installed and locked. The attacker can then leverage this to brute force the user's credentials, ultimately granting access to the crypto wallet of the victim user.

**Steps to reproduce:**

1. Open the DeBank Rabby Wallet application on either iOS or Android and complete the initial installation.
2. Lock the app by closing it entirely.
3. Attempt to unlock the app by entering an incorrect password 10 times in a row, for example, then unlock it again using the correct password.
4. Observe that blocking or throttling has not been established, which would otherwise prevent malicious activities such as brute-force attacks.

To mitigate this issue, Cure53 suggests incorporating rate limiting to the password unlock process, which should ensure that attackers are discouraged from performing malicious activities if they are able to obtain access to a device with the DeBank Rabby Wallet installed. This could be carried out by consulting other secure messaging applications, such as Threema. The *setSystemScreenLock* method from the Threema sources[11] can be adopted for usage on Android, for instance.

[11] https://github.com/threema-ch/threema-android/blob/main/app/src/main/java/ch/[...]/[...]/[...]/Set[...]#L425

**Dr.-Ing. Mario Heiderich, Cure53**
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

### RBY-01-004 WP1-WP2: Weak auto-lock time default value *(Low)*

While dynamically testing the iOS and Android DeBank Rabby Wallet applications, it was noted that the default lock time is set to 24 hours, which essentially represents the duration of user inactivity before the wallet locks again. From a security standpoint, this excessively long default lock time is problematic for an app of this nature. In the event of device theft, for instance, an attacker with access to the victim's phone could still access the wallet before it automatically locks, significantly increasing the risk of unauthorized access.

To mitigate this issue, Cure53 suggests modifying the default settings by reducing the auto-lock time to a much shorter duration, such as 5 or 10 minutes. This adjustment would enhance security by minimizing the window of opportunity for unauthorized access in the event a device is compromised. Cure53 also recommends notifying users if they keep the app running unlocked in the background, which will help to avoid the situation whereby they are unaware that their wallet has remained in an unlocked state.

### RBY-01-005 WP1-WP2: Insufficient private key password complexity check *(Info)*

While testing the iOS and Android DeBank Rabby Wallet applications, the discovery was made that the password complexity check used for protecting the wallet (and thus the user's private key) is insufficient. The implementation only enforces user passwords comprising 8 characters, allowing and accepting weak passwords such as *12345678*.

**Steps to reproduce:**

1. Open the DeBank Rabby Wallet application on either iOS or Android.
2. During the initial installation and generation of a new wallet, the user will be prompted to insert a password that is used for encrypting or decrypting their vault.
3. Enter the password *12345678* and observe that it is accepted by the application.

To mitigate this issue, Cure53 recommends hardening the password complexity validation. This is necessary to prevent users from setting weak passwords that can be easily brute-forced or guessed.

The revised password policy should comply with all requirements listed next:

- In general, longer passwords are generally more resilient against brute-force attacks, while shorter passwords of any complexity (e.g., 8 characters or below like *12345678*) are considered weak.
- The usage of different characters should be enforced, with each password requiring:
  - Lower and uppercase characters;
  - At least one digit;
  - Special character(s);
  - Prohibition of using dictionary words or words found in user-information (e.g., names);

**Dr.-Ing. Mario Heiderich, Cure53**
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

- Prohibition of using the same password twice and/or part of another password in the new password.

### RBY-01-006 WP1: Outdated iOS version support *(Info)*

***Fix note:*** *This issue has been addressed by the development team and the fix has been verified by Cure53. The described problem no longer exists.*

The application sets the minimum iOS version for the app to run at iOS 13, which potentially exposes the app to the risks of known vulnerabilities in outdated software. For instance, iOS 14 and below offer numerous publicly available exploits for privilege escalation vulnerabilities that allow a local attacker to gain root access on an iPhone.

The Apple AppStore statistics[12] for iOS indicate that, as of May 2023, the majority of users operate on iOS 16 (~81%) and iOS 15 (13%). Conclusively, this renders support for iOS 14 and below only necessary in edge-case scenarios, for which legacy software cannot be avoided for compatibility reasons.

**Affected file:**
*Info.plist*

**Affected code:**
```
<key>MinimumOSVersion</key>
<string>13.0</string>
```

To mitigate this issue, Cure53 suggests increasing the application's *minOS* version to at least iOS 15 to ensure known iOS security vulnerabilities are avoided. In addition, it is advised to closely observe the aforementioned usage statistics regarding iOS versions and adjust the *minOS* parameter accordingly. Ideally, the developer team should select a version that finds the middle ground between greatest volume of security patches applied and largest user base.

[12] https://developer.apple.com/support/app-store/

**Dr.-Ing. Mario Heiderich, Cure53**
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

## RBY-01-007 WP2: Unmaintained Android ver. support via minSDK level 24 *(Info)*

***Fix note:*** *This issue has been addressed by the development team and the fix has been verified by Cure53. The described problem no longer exists.*

While analyzing the Android Manifest contained within the APK binary, the discovery was made that the app supports Android 7 (API level 24) and upward. This can incur detrimental security implications for the app, due to the fact that Android 10 (API level 29) received its final security updates in February 2023 and is no longer actively maintained.

This increases the potential attack surface posed by the outdated environment in which the app operates. For instance, vulnerabilities such as *CVE-2019-2215*[13] and *StrandHogg 2.0*[14] can still affect Android versions up to Android 9 (API level 28).

**Affected file:**
*AndroidManifest.xml*

**Affected code:**
```
<uses-sdk android:minSdkVersion="24" android:targetSdkVersion="34" />
```

To mitigate this issue, Cure53 advises raising the minSDK level to 30 to ensure that the app can only run on an Android version that regularly receives security updates and is actively maintained. Increasing the API level in this way would reduce the potential attack surface to the app posed by known vulnerabilities in the Android version it operates on.

## RBY-01-008 WP1-WP2: App operable without password lock *(Info)*

While evaluating the DeBank Rabby Wallet's password protection feature, Cure53 noted that the app requires the user to set a password during initial installation or setup. However, the user can later remove this password entirely. This creates a vulnerability window whereby the app operates without password protection. During this period, an attacker with physical access to the device or a privileged malicious app could exploit this by setting a new password, potentially gaining unauthorized access to the user's wallet.

Albeit, backup access to the private key and mnemonic remains blocked until the user sets a new password. Nevertheless, an attacker could also cause disruptive actions and delete the wallet or address, for instance, thus removing funds in the event that the user does not sufficiently backup the mnemonic.

To mitigate this potential risk, Cure53 suggests blocking the removal of password protection entirely. Users should only be permitted to alter the password, ensuring that a password is always required for access.

[13] https://nvd.nist.gov/vuln/detail/CVE-2019-2215
[14] https://www.helpnetsecurity.com/2020/05/28/cve-2020-0096/

**Dr.-Ing. Mario Heiderich, Cure53**
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

## RBY-01-009 WP1-WP2: Vulnerable libraries in multiple components *(Info)*

During the security assessment, the observation was made that several software packages leveraged outdated versions that are vulnerable to a host of security risks. The following software packages were identified as out-of-date and potentially insecure. Notably, the version information provided is based on data collected at the time of testing. Whether these vulnerabilities are exploitable entirely depends on how the relevant functionality is used in the targeted application at present.

**Command:**
$ **snyk test --all-projects**

**Tested 115 dependencies for known issues, found 5 issues, 50 vulnerable paths.**

```
Issues with no direct upgrade or patch:
  ✗ Uncontrolled Resource Consumption [Medium
Severity][https://security.snyk.io/vuln/SNYK-RUBY-REXML-6861566] in
rexml@3.2.6
    introduced by fastlane@2.217.0 > CFPropertyList@3.0.6 > rexml@3.2.6 and
9 other path(s)
  This issue was fixed in versions: 3.2.7
  ✗ Denial of Service (DoS) [Medium
Severity][https://security.snyk.io/vuln/SNYK-RUBY-REXML-7462086] in
rexml@3.2.6
    introduced by fastlane@2.217.0 > CFPropertyList@3.0.6 > rexml@3.2.6 and
9 other path(s)
  This issue was fixed in versions: 3.3.2
  ✗ Uncontrolled Resource Consumption ('Resource Exhaustion') [Medium
Severity][https://security.snyk.io/vuln/SNYK-RUBY-REXML-7577227] in
rexml@3.2.6
    introduced by fastlane@2.217.0 > CFPropertyList@3.0.6 > rexml@3.2.6 and
9 other path(s)
  This issue was fixed in versions: 3.3.3
  ✗ Denial of Service (DoS) [Medium
Severity][https://security.snyk.io/vuln/SNYK-RUBY-REXML-7577228] in
rexml@3.2.6
    introduced by fastlane@2.217.0 > CFPropertyList@3.0.6 > rexml@3.2.6 and
9 other path(s)
  This issue was fixed in versions: 3.3.3
  ✗ Improper Restriction of XML External Entity Reference ('XXE') [High
Severity][https://security.snyk.io/vuln/SNYK-RUBY-REXML-7814166] in
rexml@3.2.6
    introduced by fastlane@2.217.0 > CFPropertyList@3.0.6 > rexml@3.2.6 and
9 other path(s)
  This issue was fixed in versions: 3.3.6

Package manager:   rubygems
```

**Dr.-Ing. Mario Heiderich, Cure53**
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

```
Target file:        apps/mobile/Gemfile.lock
```

Notably, the testing team was unable to comprehensively prove any potential impact during the limited time frame granted for this review. As such, the wider implications remain unknown at this point and should be subjected to internal research at the earliest possible convenience for the in-house team.

Generally speaking, robust supply chain security can be considerably challenging to provide to an optimal standard. Oftentimes, an easy or comprehensive solution cannot be offered, while the results and efficacy of the selected protection framework can vary depending on the integrated version of the deployed libraries.

To mitigate the existing issues as effectively as possible, Cure53 recommends upgrading all affected libraries and establishing a policy to ensure libraries remain up-to-date moving forward. This will ensure that the premise can benefit from patches rolled out for previously detected weaknesses across a variety of different solutions.

## RBY-01-011 WP1: Incomplete iOS filesystem safeguarding *(Low)*

The Cure53 testers noted that the iOS app does not take full advantage of the native iOS filesystem protections and fails to comprehensively protect some of its data files at rest. The affected files are only safeguarded until the user authenticates for the first time after booting the phone (*NSFileProtectionCompleteUntilFirstUserAuthentication*[15]). The key to decrypt these files will remain readable in memory, even while the device is locked.

To exploit this flaw, an adversary would require physical access to an iDevice set to a locked screen and a method of accessing the local storage, for instance through an SSH connection established via a jailbreak.

**Steps to reproduce:**

1. Establish an SSH connection to a jailbroken iOS testing device and navigate into the app folder. The app folder can be determined by using the objection[16] utility and running the *env* command.
2. Lock the device and run the following command, demonstrating that all files remain accessible:

   **Command:**

   ```
   iPhone-X:/var/mobile/Containers/Data/Application/EEDF357E-E758-4215-
   AFCD-514AF6E4F7ED root# tar cvf files_locked.tar.gz *

   Documents/
   Documents/mmkv/
   Documents/mmkv/mmkv.keychain
   ```

[15] https://developer.apple.com/documentation/foundation/nsfileprotectioncomple[...]tication
[16] https://github.com/sensepost/objection

**Dr.-Ing. Mario Heiderich, Cure53**
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

```
Documents/mmkv/mmkv.default
Documents/mmkv/mmkv.default.crc
Documents/mmkv/mmkv.chains
Documents/mmkv/mmkv.chains.crc
Documents/mmkv/mmkv.keyring
Documents/mmkv/mmkv.keychain.crc
Documents/mmkv/mmkv.keyring.crc
[...]
```

To mitigate this issue, it is recommended to implement the *NSFileProtection-Complete* entitlement at the application level[17] for all files, which will nullify the associated file access risks.

### RBY-01-015 WP1-WP2: Pending transactions clearable without password *(Info)*

Cure53's dynamic testing procedures confirmed that the app enforces password entry for certain actions, such as viewing the seed phrase, accessing the private key, or deleting the user's wallet. However, some actions are not protected by a password prompt, including clearing pending transactions, allowing them to be executed without entering the password.

The app remains unlocked if the auto-lock timer has not expired (default set to 24 hours) and multiple bypass methods are available that would allow a local attacker or malicious app to prevent the auto-lock mechanism from triggering. With this in mind, it is recommended to enforce a password prompt when clearing pending transactions as an additional defense-in-depth hardening measure.

[17] https://developer.apple.com/documentation/foundation/nsfileprotectioncomplete

**Dr.-Ing. Mario Heiderich, Cure53**
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

## Conclusions

This engagement marks the first security evaluation of the DeBank Rabby Wallet mobile application, which is a non-custodial EVM wallet.

Prior to initiating the assessment, the DeBank team provided several resources, including source code, documentation, and mobile app builds, facilitating a hindrance-free kickoff. Account provision was not required since the testers were able to generate or import seed phrases directly.

Cure53 maintained ongoing communication with the DeBank team via a private Slack channel. The interactions were highly effective and assistance was readily available upon request. Additionally, the testing team regularly provided updates on the project's status.

The adopted white-box approach served to maximize the breadth and depth of the test team’s analyses, allowing Cure53 to check the mobile applications for security vulnerabilities in the code and running environments. In general, the review of the mobile applications employed typical strategies to search for issues related to OWASP Mobile Top 10, such as improper platform usage, insecure data storage and communication, insufficient cryptography, and similar.

The primary concern highlighted by DeBank was the secure management of user private keys and mnemonics. It is important to ensure these sensitive assets are always encrypted and stored within the Rabby Wallet without ever being transmitted over the network or accessed by remote or malicious applications running on the same device (even with elevated privileges). Notably, the REST API was excluded from the audit scope, while Cure53 verified that seed phrases and private keys were not transmitted via requests sent towards the backend API. This validation was facilitated via a test environment that enabled traffic inspection through the circumvention of SSL certificate pinning.

Notably, the integration of hardware wallets (such as Ledger, Keystone, and OneKey), as well as third-party mobile wallet apps was not exhaustively probed, since the review team needed to hone in on other key priorities.

The mobile application is built using the React Native framework, enabling efficient development and streamlined maintenance for both iOS and Android platforms. The Android and iOS apps lack several security measures to protect against compromises, which unnecessarily increases their respective attack surface and raises concerns, as detailed below.

**Dr.-Ing. Mario Heiderich, Cure53**
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Test environment configuration was straightforward since the DeBank Rabby Wallet app can be installed on a rooted or jailbroken device, which allowed the testing team to utilize the objection framework and establish a setup that permitted inspecting traffic between the app and backend. While the lack of rooted or jailbroken device detection is subpar from a security perspective, Cure53 did not consider it a reportable weakness since it is a deliberate business decision asserted by DeBank.

The general attack surface of the Android application was examined during the initial phase. The testers were particularly interested in the application's compatibility with the respective ecosystem and platform API communication handling. Here, the exposed activities, broadcasts, content providers, and services were investigated for manipulation via intents and to ascertain the likelihood of data leakage.

The Rabby Wallet is secured with a password; however, a review of this protection layer identified several vulnerabilities. The most critical issues, particularly in the key focus areas defined by DeBank, relate to the ability to retrieve the mnemonics and password used to unlock the wallet from the process memory, even after the device is locked. This type of attack could be executed by a privileged malicious application or adversary running alongside the DeBank Rabby Wallet on the user's device, as indicated in RBY-01-001 and RBY-01-002.

The source code audit confirmed that the DeBank developers have paid careful attention to encrypting the passphrase and mnemonics at rest. However, during execution, both the passphrase and mnemonics are passed in plaintext through the application context. To effectively protect these sensitive secrets from privileged attackers, performant safeguards must be implemented for handling this data during execution.

The user's wallet automatically locks after a predefined period. However, it was discovered that the default timeout of 24 hours is excessive, as discussed in RBY-01-004. Additionally, an attacker under the guise of a malicious app could bypass this timer altogether by manipulating the system time (see RBY-01-010).

The Rabby Wallet mobile applications rely on a hardcoded secret, which was directly embedded into the installer files at the time of testing. Consultations with the client verified that the secret plays a crucial role in generating cryptographic values for a user's wallet. However, the secret is easily extractable in clear text with minimal effort, even for attackers with limited experience, as it is shared across all installations of the Rabby Wallet application (see RBY-01-012).

In general, the current design and implementation of the user password verification protocols, which serve to unlock the wallet and gain access to the private key, are susceptible to several weaknesses. These are outlined below:

**Dr.-Ing. Mario Heiderich, Cure53**
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

- The backup process for sensitive information, such as the seed phrase or private key, requires the user to enter their password. However, this password validation can be bypassed by a privileged malicious app, as the verification process can be easily hooked and patched out (RBY-01-013).

- Additionally, the overall password verification process is flawed since it lacks any form of rate limiting, which allows a malicious app to brute-force the password (RBY-01-003).

- Furthermore, the password complexity for the wallet is weak as simple compositions are permitted, with a minimum length of 8 characters representing the only enforced requirement (RBY-01-005).

- Although most sensitive actions within the mobile application are protected by an additional password prompt, pending transactions can be cleared without requiring the user to re-enter their password (RBY-01-015).

- The Android version could be further improved by removing support for outdated and unmaintained versions facilitating n-day vulnerabilities (see RBY-01-007). Additionally, the iOS version could be fortified by removing support for old iOS versions (see RBY-01-006).

- To end on a positive note, the assessors verified that sensitive information was not stored in clear text on the device, while the applications were not vulnerable to common mobile attack vectors. For example, it was not possible to remotely crash the Android application by sending malicious intents.

Moving forward, Cure53 highly recommends continuously reevaluating the scope's security performance through recurring audits. These should be conducted at least once a year or in the event of major modification, hence guaranteeing that the construct remains protected against emerging threats and has not introduced bugs via new implementations.

To conclude, this Q3 2024 security review achieved satisfactory coverage over all WPs, identifying a multitude of security-relevant limitations that provide ample opportunity for hardening measures. Once the existing faults have been addressed, the DeBank Rabby Wallet mobile application will exert a stronger security posture suitable for production use.

Cure53 would like to thank Dexin Yuan, Guoyan Yang, and Lei Chen from the DeBank team for their excellent project coordination, support, and assistance, both before and during this assignment.