Pontem Mobile Wallet

Audit

Presented by:

OtterSec contact@osec.io

Maher Azzouzi maher@osec.io

Naveen Kumar J naina@osec.io

Robert Chen r@osec.io

Contents

01 Executive Summary **2**

Overview 2

Key Findings 2

02 Scope **3**

03 Findings **4**

04 Vulnerabilities **5**

OS-PMW-ADV-00 [high] [resolved] | React Keychain Weak Access Control 6

OS-PMW-ADV-01 [med] [resolved] | Sensitive Data Exposure Via Logs And Events 7

OS-PMW-ADV-02 [med] [resolved] | Vulnerable NPM Packages 8

OS-PMW-ADV-03 [low] [resolved] | Mnemonic Verification Bypass 9

OS-PMW-ADV-04 [low] [resolved] | Missing Null IV Check 10

OS-PMW-ADV-05 [low] [resolved] | Not Suitable CPRNG Used To Generate Salt 11

OS-PMW-ADV-06 [low] [resolved] | Missing Dependencies Exact Version Pinning 12

OS-PMW-ADV-07 [low] [resolved] | Permissive Keychain.ACCESSIBLE Enum 13

05 General Findings **14**

OS-PMW-SUG-00 | PBKDF2 Insufficient Key Rotations 15

OS-PMW-SUG-01 | Lack Of Root Detection In Pontem Wallet App 16

OS-PMW-SUG-02 | WipePrivateData() Method Is Not Used 17

OS-PMW-SUG-03 | Weak Password Constraints During Wallet Creation 18

OS-PMW-SUG-04 | Potential Private Key Disclosure 19

OS-PMW-SUG-05 | Possible Use Of Clear Text Traffic 20

OS-PMW-SUG-06 | Improve Browser Input Sanitization 21

Appendices

A Vulnerability Rating Scale **22**

B Procedure **23**

01 | Executive Summary

Overview

Pontem Network Mobile Wallet engaged OtterSec to perform an assessment of the `mobile-wallet` program. This assessment of the source code was conducted between January 16th and February 3rd, 2023 by 3 engineers. For more information on our auditing methodology, see Appendix B.

Critical vulnerabilities were communicated to the team prior to the delivery of the report to speed up remediation. After delivering our audit report, we worked closely with the team to streamline patches and confirm remediation. We delivered final confirmation of the patches February 10th, 2023.

Key Findings

Over the course of this audit engagement, we produced 15 findings total.

In particular, we reported security misconfigurations around keyrings (OS-PMW-ADV-00), user credentials and vault data exposure (OS-PMW-ADV-01), issues around dependencies (OS-PMW-ADV-02), and various other mobile attack vectors.

We also made recommendations around improving encryption standards (OS-PMW-SUG-00, OS-PMW-SUG-05), sensitive data handling (OS-PMW-SUG-01), and various defense in-depth mitigations.

Overall, we commend the Pontem Network Mobile Wallet team for being responsive and knowledgeable throughout the audit.

02 | Scope

The source code was delivered to us in a git repository at github.com/pontem-network/mobile-wallet. This audit was performed against commit e678a44.

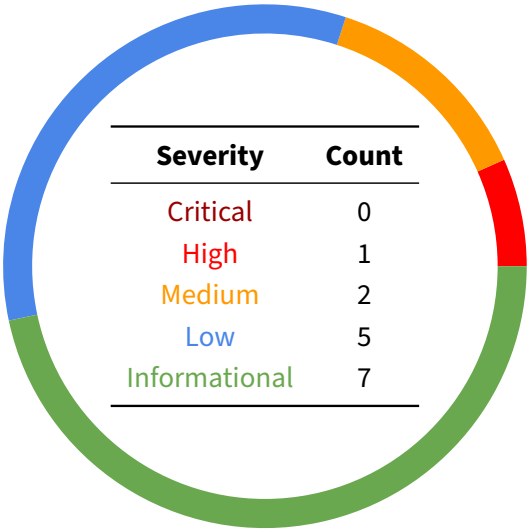
A brief description of the programs is as follows.

Name	Description
mobile-wallet	A non-custodial cryptocurrency wallet built on Aptos for mobile users.

03 | Findings

Overall, we report 15 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings don't have an immediate impact but will help mitigate future vulnerabilities.



Severity	Count
Critical	0
High	1
Medium	2
Low	5
Informational	7

04 | Vulnerabilities

Here we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

ID	Severity	Status	Description
OS-PMW-ADV-00	High	Resolved	React-native-keychain has no access control while accessing data by default.
OS-PMW-ADV-01	Medium	Resolved	User credentials and vault data are exposed through logs and events.
OS-PMW-ADV-02	Medium	Resolved	NPM audit detected 15 vulnerabilities in the packages.
OS-PMW-ADV-03	Low	Resolved	The seed phrase verification bypass with `isSetupCompleted` is set to true by default.
OS-PMW-ADV-04	Low	Resolved	`Aes.randomKey` used to generate IV, which may return nil in some cases.
OS-PMW-ADV-05	Low	Resolved	Use `generateKey()` in favor of `getRandomValues()`.
OS-PMW-ADV-06	Low	Resolved	Dependencies are not pinned with exact versions, which opens a ”malicious dependencies” attack surface.
OS-PMW-ADV-07	Low	Resolved	Keychain.ACCESSIBLE by default allows data to be accessed when the device is unlocked.

OS-PMW-ADV-00 [high] [resolved] | React Keychain Weak Access Control

Description

Pontem uses the `setGenericPassword` function from the `react-native-keychain` library to store the user's password and address in the device. This function has different options, one being that `Keychain.ACCESS_CONTROL` which has many access control choices that require a password or touch ID to access the keychain data. However, by default, there is no access control while accessing the keychain data.

react-native-keychain/../oblador/keychain/KeychainModule.java JAVA

```java
/** Get access control value from options or fallback to {@link
    ↪ AccessControl#NONE}. */
@AccessControl
@NonNull
private static String getAccessControlOrDefault(@Nullable final
    ↪ ReadableMap options) {
    return getAccessControlOrDefault(options, AccessControl.NONE);
}
```

Remediation

It is recommended to specify the desired access control from available options.

JS

```
await setGenericPassword(address, password, {
        accessControl : <SUITABLE OPTION>
    });
```

Patch

Patch added in commit 62d7365.

src/components/auth/importAccount/form/index.js DIFF

```
-   await setGenericPassword(address, password);
+   await setGenericPassword(address, password, {
+      accessControl:
    ↪  Keychain.ACCESS_CONTROL.BIOMETRY_ANY_OR_DEVICE_PASSCODE,
+    });
```

OS-PMW-ADV-01 [med] [resolved] | Sensitive Data Exposure Via Logs And Events

Description

In some places, the app logs and emits events with user-sensitive information. Logging levels must be set appropriately before the product ships so that sensitive user data cannot mistakenly be exposed to potential attackers. Attackers may make use of these logs and events to gain control over the user's wallet.

JS

```
const storeCredentials = async ({address, password}) => {
    console.log('account', address, password);
    await setGenericPassword(address, password);
};

protected async _updateVault(password: string | undefined =
    ↪  this.password) {
    await this._updateInternalVault(password);
    this.emit('update', this.state.vault);
}
```

Remediation

Avoid user-sensitive data exposure through logs, events, or any error messages.

Patch

Patch added to removed logs and events in commit f19c7fd.

OS-PMW-ADV-02 [med] [resolved] | Vulnerable NPM Packages

Description

A total of 15 vulnerabilities were identified with `npm audit`. These include `Command Injection`, `Prototype Pollution`, `Denial of Service`, and `Sensitive Information Exposure` vulnerabilities. These vulnerabilities may pose a significant risk to the security of the app.

BASH

```
ios git:(master) ✗ npm audit
# npm audit report

lodash  <=4.17.20
Severity: critical
Prototype Pollution in lodash
Regular Expression Denial of Service (ReDoS) in lodash
Command Injection in lodash

json5  <1.0.2 || >=2.0.0 <2.2.2
Severity: high
Prototype Pollution in JSON5 via Parse Method

node-fetch  <=2.6.6
Severity: high
Vulnerable to Exposure of Sensitive Information to an Unauthorized Actor

semver  <4.3.2
Severity: high
Regular Expression Denial of Service in semver
[..]

15 vulnerabilities (1 moderate, 13 high, 1 critical)
```

Remediation

It is recommended that the package and its dependencies be updated to the latest version to address these vulnerabilities. It is also important to monitor the package for any future security updates and apply them promptly. Regular security assessments should also be conducted to identify and address any new vulnerabilities that may arise.

Patch

Packages are updated in commit 8f73fb4.

OS-PMW-ADV-03 [low] [resolved] | Mnemonic Verification Bypass

Description

When creating a wallet after making a new password, the user will be asked to note down their secret words (i.e., mnemonic), which are essential for wallet recovery. But, this step can be skipped as the user will be navigated to the dashboard if the user attempts to close and reopen the app.

src/components/_hooks/useKeyring.ts TS

```
export const useKeyring = () => {
  [..]

  const [isSetupCompleted, setIsSetupCompleted] =
      ↪  useState<boolean>(true);
  [..]

  const controller = useMemo(() => {
    const {vault, setupCompleted} = keyringSetup;
    const ctrl = new KeyringController({
      initState: {
        vault,
      },
    });
}
```

The default value of `isSetupCompleted` is set to `true` instead of `false`. Because of this, if the scenario presented above occurs, the user will be asked to unlock the wallet and later be navigated to the dashboard screen without knowing their `mnemonic`, which is essential for recovery.

Remediation

Use default value to be `false` for `isSetupCompleted`.

TS

```
const [isSetupCompleted, setIsSetupCompleted] = useState<boolean>(false);
```

Patch

Updated logic in commit 73d8afe.

OS-PMW-ADV-04 [low] [resolved] | Missing Null IV Check

Description

The underlying implementation of the function `randomKey` can sometimes return nil. This means that in some cases, a zero or empty IV will be returned.

react-native-aes/../ios/RCTAes/lib/AesCrypt.m OBJECTIVE-C

```
+ (NSString *) randomKey: (NSInteger)length {
    [..]
    if (result != noErr) {
        return nil;
    }
    return [self toHex:data];
}
```

The `Encryptor` class uses the function `Aes.randomKey` to generate IV, which will be used in encryption and decryption.

src/packages/keyrings/Encryptor.js JS

```
  _encryptWithKey = async (text, keyBase64) => {
    const iv = await Aes.randomKey(16);
    [..]
  };
```

Using a null initialization vector to encrypt the same data with the same key would produce the same ciphertext. This is potentially vulnerable to a dictionary attack, where an attacker could determine the plaintext that produced a given ciphertext.

Remediation

To prevent this, it is recommended to handle the nil case when using the function `Aes.randomKey`.

Patch

Handled null case in commit 5c4fb61.

OS-PMW-ADV-05 [low] [resolved] | Not Suitable CPRNG Used To Generate Salt

Description

Pontem `Encryptor` class uses `crypto.getRandomValues` function to generate salt. This function is not guaranteed to run in a secure context and is not recommended for generating encryption keys.

src/packages/keyrings/Encryptor.js JS

```
  _generateSalt(byteCount = 32) {
    const view = new Uint8Array(byteCount);
    global.crypto.getRandomValues(view);
    // eslint-disable-next-line no-undef
   [..]
  }
```

The documentation of `getRandomValues` function says:

```
Don't use getRandomValues() to generate encryption keys.
Instead, use the generateKey() method.
```

Remediation

As per the documentation, use the `generateKey()` method to generate encryption keys.

Patch

Patch added in commit 5c4fb61.

OS-PMW-ADV-06 [low] [resolved] | Missing Dependencies Exact Version Pinning

Description

The app uses a significant amount of dependencies. Most of these dependencies are not pinned to an exact version; most are set like: `(^X.X.X)`, which will receive the most recent major version. This could enable dependency attacks.

Remediation

To remediate the problem, pin dependencies to an exact version to minimize the possibility of using a malicious version.

Patch

Resolved in commit fa8ac65.

OS-PMW-ADV-07 [low] [resolved] | Permissive Keychain.ACCESSIBLE Enum

Description

The enum `Keychain.ACCESSIBLE` specifies when a keychain item is accessible. The default value is `WHEN_UNLOCKED`, which indicates that the data in the keychain item can be accessed when the device is unlocked.

Remediation

It is recommended to use `WHEN_UNLOCKED_THIS_DEVICE_ONLY`. This will additionally aid in keeping the items on the same device, not allowing migrating items to a new device.

TS

```ts
await setGenericPassword(address, password, {
        accessible: Keychain.ACCESSIBLE.WHEN_UNLOCKED_THIS_DEVICE_ONLY,
    });
```

Patch

Added in commit 654e191.

src/components/auth/importAccount/form/index.js DIFF

```
    await setGenericPassword(address, password, {
       accessControl:
     ↪  Keychain.ACCESS_CONTROL.BIOMETRY_ANY_OR_DEVICE_PASSCODE,
+      accessible: Keychain.ACCESSIBLE.WHEN_UNLOCKED_THIS_DEVICE_ONLY,
    });
```

05 | General Findings

Here we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent antipatterns and could lead to security issues in the future.

ID	Description
OS-PMW-SUG-00	PBKDF2 insufficient key rotations may be vulnerable to cracking due to the low number of rotations.
OS-PMW-SUG-01	The app does not check if the device it is running on is rooted or jailbroken.
OS-PMW-SUG-02	The wipePrivateData method is defined in the class HDKey but is not called.
OS-PMW-SUG-03	The current implementation of the app accepts very weak passwords during the creation of a wallet.
OS-PMW-SUG-04	A successful brute-force attack could result in the unauthorized disclosure of the private key.
OS-PMW-SUG-05	Android:usesCleartextTraffic=”false” should be added in the app’s manifest.
OS-PMW-SUG-06	Browser input sanitization is not enough to prevent unintended actions.

OS-PMW-SUG-00 | PBKDF2 Insufficient Key Rotations

Description

Usually, a strong password-based KDF should protect against brute forcing, making it very long, as KDFs are designed to be slow. However, this app used PBKDF2 with only 5000 iterations. This means that the more times the PBKDF2 function is iterated, the longer it takes to compute the password hash.

src/packages/keyrings/HD/Encryptor.js JS

```
  _generateKey = (password, salt, lib) =>
    lib === 'original'
      ? Aes.pbkdf2(password, salt, 5000, 256)
      : AesForked.pbkdf2(password, salt);
```

PBKDF2 is well-known to have weaknesses and CPU and GPU optimizations for faster brute force. Tools like John-the-Ripper already provide GPU support for cracking PBKDF2-HMAC-SHA256. Using such a few rounds with PBKDF2 is a wrong choice that makes brute-forcing even easier.

Remediation

According to OWASP documentation, the following are:

```
PBKDF2-HMAC-SHA1   : 720,000 iterations.
PBKDF2-HMAC-SHA256 : 310,000 iterations.
PBKDF2-HMAC-SHA512 : 120,000 iterations.
```

OS-PMW-SUG-01 | Lack Of Root Detection In Pontem Wallet App

Description

A rooted device could allow an attacker to access sensitive information stored in the wallet or manipulate the app's functionality.

Remediation

Implement a check within the app to detect if the device is rooted and take appropriate actions based on that. Such as not running the app and prompting the user that their device is rooted.

OS-PMW-SUG-02 | WipePrivateData() Method Is Not Used

Description

This method appears to be wiping private data by using the `crypto.randomBytes()` method to generate random bytes, then by using the `Buffer.copy()` method to copy those random bytes over the data stored in the `_privateKey` property. This effectively ”overwrites” the original data stored in `_privateKey` with random data, making it unrecoverable.

src/packages/keyrings/HD/hdkey/HDKey.ts JS

```
wipePrivateData() {
    if (this._privateKey) {
    crypto.randomBytes(this._privateKey.length).copy(this._privateKey);
    }
    this._privateKey = null;
    return this;
}
```

This method should be called and executed after the private data is no longer needed. Not calling this function and leaving the private data in memory may present a security risk.

OS-PMW-SUG-03 | Weak Password Constraints During Wallet Creation

Description

The current implementation of the app accepts very weak passwords during the creation of a wallet, as the only requirement is that the password has a length of nine or more characters. This poses a security risk, as a simple and easily guessable password can be used to access the wallet.

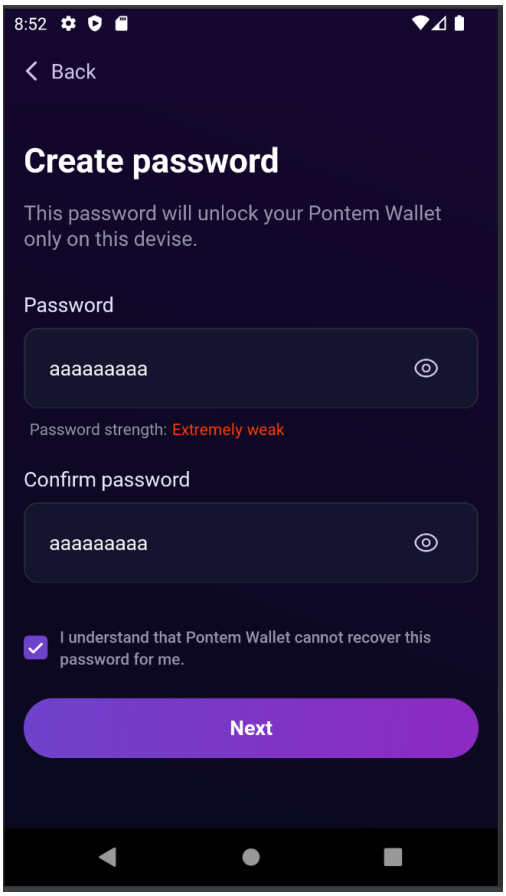


Figure 05.1: e.g., ‘aaaaaaaaa‘ is acceptable as the wallet password.

Remediation

It is recommended to enforce stronger password constraints during the wallet creation process. This can include adding requirements for using numbers and/or special characters in the password and implementing a minimum password strength policy. This will make it harder for attackers to guess or crack the password and protect the user’s assets.

OS-PMW-SUG-04 | Potential Private Key Disclosure

Description

The Pontem wallet app allows the display of private keys under ”Show Private Key” in the ”General Settings” menu. The user is prompted first to enter a password. However, this password protection is vulnerable to brute-force attacks, which could allow non-authenticated entities to gain access to a user’s private key by guessing the password.

Remediation

To mitigate this attack, the app should rate limit the number of failed attempts. A threshold of three or five attempts would be sufficient to protect the private key. Additionally, consider implementing an additional layer of protection, such as a biometric authentication, before the private key is displayed.

OS-PMW-SUG-05 | Possible Use Of Clear Text Traffic

Description

Ensuring that Pontem only uses secure network connections is important to securing the app and data in transit. Therefore, for API level 27 or lower, the app can use unencrypted network traffic, such as HTTP, instead of HTTPs.

Remediation

Setting `android:usesCleartextTraffic="false"` in the app's manifest file prevents the app from using cleartext network traffic and improves the app's security.

OS-PMW-SUG-06 | Improve Browser Input Sanitization

Description

The browser currently appears to be sanitizing URLs by only encoding quotes and replacing CRLF chars. However, imposing more restrictions by improving sanitization around user inputs may make the functionality more intended.

```ts
export const sanitizeUrlInput = (url: string) =>
  url.replace(/'/g, '%27').replace(/[\r\n]/g, '');
```

Scheme validation should be done properly. For example, this URL will execute arbitrary JavaScript and pops an alert `javascript://xyz.com/%0aalert(location.origin)`.

Remediation

It is recommended to improve sanitizing user input to prevent unwanted actions.

A | Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings can be found in the General Findings section.

Critical	Vulnerabilities that immediately lead to loss of user funds with minimal preconditions Examples: • Misconfigured authority or access control validation • Improperly designed economic incentives leading to loss of funds
High	Vulnerabilities that could lead to loss of user funds but are potentially difficult to exploit. Examples: • Loss of funds requiring specific victim interactions • Exploitation involving high capital requirement with respect to payout
Medium	Vulnerabilities that could lead to denial of service scenarios or degraded usability. Examples: • Malicious input that causes computational limit exhaustion • Forced exceptions in normal user flow
Low	Low probability vulnerabilities which could still be exploitable but require extenuating circumstances or undue risk. Examples: • Oracle manipulation with large capital requirements and multiple transactions
Informational	Best practices to mitigate future security risks. These are classified as general findings. Examples: • Explicit assertion of critical internal invariants • Improved input validation

B | Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the implementation of the program requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of sum, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to get a comprehensive understanding of the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that the other missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.