
# Motion Wallet

Security Assessment

April 3rd, 2026 — Prepared by OtterSec

Caue Obici caue@osec.io

# Table of Contents

# 01 — Executive Summary

## Overview

MoveIndustries engaged OtterSec to assess the `motion-wallet` program. This assessment was conducted between March 13th and March 19th, 2026. For more information on our auditing methodology, refer to chapter 07.

## Key Findings

In particular, we reported a vulnerability where the wallet's approval popup does not simulate transactions or show balance changes before signing, so users may authorize malicious actions without understanding their real impact, increasing exposure to phishing attacks (OS-MMV-ADV-00). Also, the wallet does not verify displayed tokens, so users may mistake scam or spoofed assets for legitimate ones, increasing the risk of interacting with malicious tokens and potentially losing funds (OS-MMV-ADV-01).

Additionally, the wallet incorrectly treats token symbols as trusted identifiers, even though symbols are untrusted and may be duplicated by malicious tokens, potentially resulting in spoofed assets to inherit legitimate pricing and misleading users (OS-MMV-ADV-02). Furthermore, the wallet may expose the recovery phrase if it is entered through normal non-password inputs, since browsers may persist such field contents to disk or keep them in memory, allowing attackers with remote or physical access to recover the mnemonic (OS-MMV-ADV-03).

We also recommended ensuring that sensitive values are passed through authenticated chrome-runtime messaging instead of reading approval context from query parameters (OS-MMV-SUG-00), and suggested displaying the full origin, including protocol, hostname, and port, during connection approval, message signing, and transaction signing flows to help detect insecure or otherwise suspicious origins before approving sensitive actions (OS-MMV-SUG-01). We further advised encrypting sensitive metadata in storage to reduce information leakage on stolen or compromised devices (OS-MMV-SUG-02).

# 02 — Scope

The source code was delivered to us in a git repository at https://github.com/MoveIndustries/motion-wallet. This audit was performed against commit 400cbff.

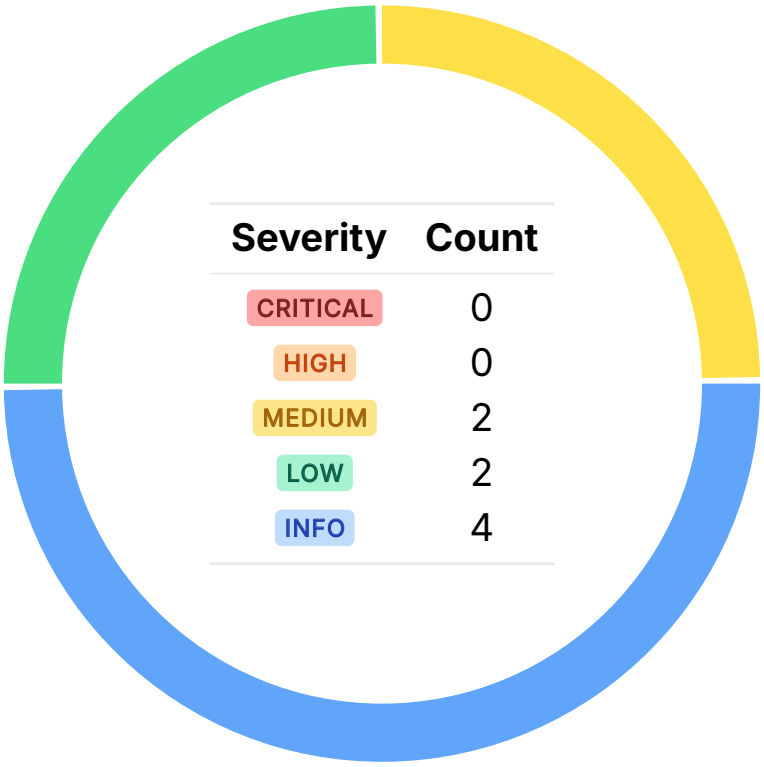
**A brief description of the program is as follows:**

| Name | Description |
| --- | --- |
| motion-wallet | A browser extension wallet for the Movement blockchain that lets users manage multiple wallets, send and receive MOVE, view balances and history, and connect to dApps. It also supports swaps, bridging from Movement to Ethereum, and testnet/mainnet switching. |

# 03 — Findings

Overall, we reported 8 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 do not have an immediate impact but will aid in mitigating future vulnerabilities.



| Severity | Count |
| --- | --- |
| CRITICAL | 0 |
| HIGH | 0 |
| MEDIUM | 2 |
| LOW | 2 |
| INFO | 4 |

# 04 — Vulnerabilities

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

Rating criteria can be found in chapter 06.

| ID | Severity | Status | Description |
|---|---|---|---|
| OS-MMV-ADV-00 | MEDIUM | RESOLVED | The wallet's approval popup does not simulate transactions or show balance changes before signing, so users may authorize malicious actions without understanding their real impact, increasing exposure to phishing attacks. |
| OS-MMV-ADV-01 | MEDIUM | RESOLVED | The wallet does not verify displayed tokens, so users may mistake scam or spoofed assets for legitimate ones, increasing the risk of interacting with malicious tokens and potentially losing funds. |
| OS-MMV-ADV-02 | LOW | RESOLVED | The wallet incorrectly treats token symbols as trusted identifiers, even though symbols are untrusted and may be duplicated by malicious tokens. This may result in spoofed assets to inherit legitimate pricing, misleading users. |
| OS-MMV-ADV-03 | LOW | RESOLVED | The wallet may expose the recovery phrase if it is entered through normal non-password inputs, since browsers may persist such field contents to disk or keep them in memory, allowing attackers with remote or physical access to recover the mnemonic. |

## Missing Transaction Simulation Preview MEDIUM OS-MMV-ADV-00

### Description

The transaction approval popup currently asks the user to authorize a transaction without first simulating its effects or presenting the expected balance deltas. As a result, the actual value of the transaction remains hidden at signing time. his increases the risk of an attacker draining the wallet via phishing, where malicious dApps disguise dangerous transactions as benign actions. Without a pre-execution preview, the wallet fails to surface suspicious or abnormal transactions that may be against the users intent.

### Remediation

Add transaction simulation and explicit balance-delta display to improve user visibility.

### Patch

Resolved in PR#24.

## Absence of Verified Token Whitelist MEDIUM OS-MMV-ADV-01

### Description

The wallet shows tokens without verifying whether they are legitimate or malicious, so scam tokens can appear indistinguishable from real assets. Attackers may exploit this by creating spoofed tokens with misleading names, symbols, or icons that trick users into trusting them. If users transfer, swap, or approve such tokens, they may be pushed into harmful interactions that may result in loss of funds.

### Remediation

Implement a verified-token whitelist and clearly mark unverified assets to reduce the risk of spoofing with scam tokens.

### Patch

Resolved in PR#23.

## Reliance on Untrusted Token Symbol LOW OS-MMV-ADV-02

### Description

The wallet utilizes `metadata.symbol` as trusted input for pricing and token-specific logic when symbol is `MOVE` or `APT`, even though symbols are untrusted and non-unique. This implies that two distinct tokens sharing the same symbol will resolve to the same price entry, even if they are unrelated assets. As a result, a malicious token may inherit the displayed price of a legitimate token simply by reusing its symbol.

>_ *src/popup/components/token/TokenList.tsx* TYPESCRIPT

```tsx
export default function TokenList({ onTokenSelect, selectedToken }: TokenListProps) {
  const { data: balances, isLoading, error } = useBalances()
  const symbols = balances?.map(b => b.metadata.symbol) ?? []
  const { data: prices } = useTokenPrices(symbols)
  [...]
  return (
    <div className="flex flex-col" style={{ gap: 10 }}>
      {balances.map((balance) => {
        const price = prices?.[balance.metadata.symbol]
        const amt = parseFloat(formatTokenAmount(balance.amount, balance.metadata.decimals))
        const usdValue = price ? amt * price.usdPrice : 0
        [...]
      })}
    </div>
  )
}
```

### Remediation

Ensure to key all token logic off a unique on-chain identifier such as the asset type or contract address.

### Patch

Resolved in PR#23.

## Recovery Phrase Input Exposure LOW OS-MMV-ADV-03

### Description

The wallet is exposed to a class of browser-extension leakage referred to as the Demonic vulnerability (CVE-2022-32969), where a secret recovery phrase entered into a normal text input may be persisted by the browser on disk, outside the wallet's intended storage boundary. In Chromium- and Firefox-based browsers, session restore mechanisms may save the contents of non-password form fields to disk in plaintext. If the wallet collects a `BIP39` mnemonic through an input field that is not treated as a protected password field, the recovery phrase may be stored unencrypted in browser session data. This will allow an attacker to retrieve it and gain access to the wallet. A related risk exists in memory as well, where the phrase may remain in memory, allowing a sufficiently privileged local attacker to extract the mnemonic phrase directly from process memory.

### Remediation

Utilize hardened input flows, avoid ordinary text fields, disable persistence features, and minimize on-disk and in-memory exposure.

### Patch

Resolved in PR#26.

# 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 anti-patterns and may result in security issues in the future.

| ID | Description |
| --- | --- |
| OS-MMV-SUG-00 | The approval popup reads sensitive request context from query parameters, which other extensions may be able to set when opening the wallet page. |
| OS-MMV-SUG-01 | The wallet shows only the hostname in approval flows, which hides important security context such as the protocol and port, rendering it harder for users to detect insecure or otherwise suspicious origins before approving sensitive actions. |
| OS-MMV-SUG-02 | The wallet stores some metadata in unencrypted storage. In an eventual system compromise (through remote or physical access), these metadata can be leaked. |
| OS-MMV-SUG-03 | The form only enforces password length, even though the configuration also requires uppercase, lowercase, and numeric characters, allowing users to set weaker passwords than intended. |

## Unsafe Passing of Approval Context OS-MMV-SUG-00

### Description

In the current implementation, the approval popup accepts `txId`, `msgId`, and origin through URL query parameters, even though other extensions may open `chrome-extension://` pages and supply arbitrary values. This allows untrusted extensions to influence security-sensitive popup state, creating a risk of request ambiguity, origin spoofing, or misleading approval user-interface. Because the approval screen is a trusted user decision surface, its context should not come from tamperable URL input.

### Remediation

Ensure that these values are passed through authenticated `chrome.runtime` messaging instead of reading approval context from query parameters.

### Patch

Resolved in PR#25.

## Incomplete Display of Origin Information OS-MMV-SUG-01

### Description

The wallet currently displays only the hostname during connection approval, message signing, and transaction signing flows. This omits important origin context, particularly the scheme ( `http` vs `https` ) and port, which are part of the actual security boundary of the requesting site. This renders it harder to determine whether the request originated from a secure or insecure connection. A malicious attacker may therefore spoof the origin through man-in-the-middle attacks, and the user may approve a sensitive action under the impression that it came from a trusted site, without realizing the page was served over plain `HTTP` or from an unusual port.

### Remediation

Display the full origin, including protocol, hostname, and port.

### Patch

Resolved in PR#25.

## Unencrypted Sensitive Metadata Storage OS-MMV-SUG-02

### Description

The wallet stores some sensitive metadata, such as wallet addresses, dApp connections, and transaction history, in unencrypted storage. Thus, in case of physical-access attacks or forensic scenarios, it may be able to recover this data even if private keys remain protected. This may leak useful information about the user's accounts, activity, counterparties, and trusted applications, enabling targeted phishing or privacy loss. The severity depends on the intended threat model, however it still meaningfully degrades the wallet's security.

### Remediation

Encrypt sensitive metadata in storage to reduce information leakage on stolen or compromised devices.

### Patch

Resolved in PR#27.

## Incomplete enforcement of password rules OS-MMV-SUG-03

### Description

The existing password policy declares uppercase ( `REQUIRE_UPPERCASE` ), lowercase ( `REQUIRE_LOWERCASE` ), and numeric requirements ( `REQUIRE_NUMBER` ), but the form only enforces minimum length ( `MIN_LENGTH` ) and password confirmation. As a result, users may set weaker passwords than intended, despite the configuration suggesting stronger protection is in place, creating a mismatch between the declared and actual password policy.

*>_ src/onboarding/components/PasswordForm.tsx* TYPESCRIPT

```
export default function PasswordForm({ onSubmit, loading, submitLabel = 'Continue' }:
    ↪  PasswordFormProps) {
  const { password, confirmPassword, agreedToTerms, setPassword, setConfirmPassword,
      ↪  setAgreedToTerms } = useOnboarding()
  const [showPassword, setShowPassword] = useState(false)
  const [showConfirm, setShowConfirm] = useState(false)

  const passwordError = password.length > 0 && password.length < PASSWORD_CONFIG.MIN_LENGTH
    ? `At least ${PASSWORD_CONFIG.MIN_LENGTH} characters`
    : null

  const confirmError = confirmPassword.length > 0 && password !== confirmPassword
    ? 'Passwords do not match'
    : null

  const isValid = password.length >= PASSWORD_CONFIG.MIN_LENGTH
    && password === confirmPassword
    && agreedToTerms

  const handleSubmit = (e: React.FormEvent) => {
    e.preventDefault()
    if (isValid && !loading) onSubmit()
  }
  [...]
}
```

### Remediation

Ensure the password validation also enforce `REQUIRE_UPPERCASE` , `REQUIRE_LOWERCASE` and `REQUIRE_NUMBER` .

### Patch

Resolved in PR#22.

# 06 — Vulnerability Rating Scale

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

**CRITICAL**

Vulnerabilities that immediately result in a 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 may result in a 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 may result in denial of service scenarios or degraded usability.

Examples:

- Computational limit exhaustion through malicious input.
- Forced exceptions in the normal user flow.

**LOW**

Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk.

Examples:

- Oracle manipulation with large capital requirements and multiple transactions.

**INFO**

Best practices to mitigate future security risks. These are classified as general findings.

Examples:

- Explicit assertion of critical internal invariants.
- Improved input validation.

# 07 — 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 program's implementation 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 thumb, 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 comprehensively understand 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 others may have 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.