# Kucoin Extension Wallet - Client Side

Security Assessment

CertiK Assessed on Jan 14th, 2026

CertiK Assessed on Jan 14th, 2026

# Kucoin Extension Wallet - Client Side

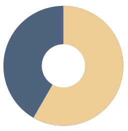
The security assessment was prepared by CertiK.

## Executive Summary

| TYPES | PLATFORM | METHODS |
|---|---|---|
| Wallet | Browser Extension | Dynamic Testing, Manual Review, Testnet Deployment |

| LANGUAGE | TIMELINE |
|---|---|
| TypeScript | Preliminary comments published on 12/31/2025<br>Final report published on 01/14/2026 |

## Vulnerability Summary



| 12 | 12 | 0 | 0 | 0 |
|---|---|---|---|---|
| Total Findings | Resolved | Partially Resolved | Acknowledged | Declined |

| | | | | |
|---|---|---|---|---|
| 🟥 | 0 | Critical | | Critical risks are those that impact the safe functioning of a platform and must be addressed immediately. Users should be cautious when interacting with any application with outstanding critical risks. |
| 🟧 | 0 | High | | High risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds, thief of user data, and/or loss control of the application. |
| 🟨 | 0 | Medium | | Medium risks may not pose a security risk at a large scale, but they can affect the overall functioning of a platform or be used to target a certain group of users. |
| 🟨 | 7 | Low | 7 Resolved | Low risks can be any of the above, but on a smaller impact. They generally do not compromise the overall integrity of the project. |
| 🟦 | 5 | Informational | 5 Resolved | Informational errors are often recommendations to improve the configuration or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the application. |

# TABLE OF CONTENTS | KUCOIN EXTENSION WALLET - CLIENT SIDE

## SCOPE | KUCOIN EXTENSION WALLET - CLIENT SIDE

| | |
|---|---|
| Extension(Md5) | 369bd5935747d4e4a2a3241c013efa578d1a58b8 |
| Page Provider(Md5) | a44626f236d0145c19f5ced69b557c11ca86ac5a |

## APPROACH & METHODS | KUCOIN EXTENSION WALLET - CLIENT SIDE

This report has been prepared for Kucoin to discover issues and vulnerabilities in the application of the Kucoin Extension Wallet - Client Side project. Kucoin wallet extension is a multi-chain browser wallet that allows users to manage crypto assets, interact with decentralized applications (DApps), and explore the Web3 ecosystem.

The pentest was a manual assessment of the security of the application's functionality, business logic, and vulnerabilities, such as those cataloged in the OWASP Top 10. The assessment also included a review of security controls and requirements listed in the OWASP Application Security Verification Standard (ASVS). The pentesters leveraged tools to facilitate their work. However, the majority of the assessment involved manual analysis.

The main objective of the engagement is to test the overall resiliency of the application to various real-world attacks against the application's controls and functions and thereby be able to identify its weaknesses and provide recommendations to fix and improve its overall security posture.

Two members of the CertiK team were involved in completing the engagement, which took place over the course of 8 days in December 2025 and yielded 12 security-relevant findings. The most significant vulnerability is Dapp blacklist bypass.

Other weaknesses were also found and are detailed in the Findings section of the report. We recommend addressing these findings to ensure a high level of security standards and industry practices and to raise the security posture of the application.

# REVIEW NOTES | KUCOIN EXTENSION WALLET - CLIENT SIDE

The assessment focuses on evaluating the security of the KuCoin wallet browser extension. The auditors prepared a list of threat hypotheses to guide them through the source code review and dynamic testing process.

**Threat hypothesis for general wallet security**

- How does the application generate the seed phrase and private key?
- How and where does the application store the seed phrase and private key?
- Does the wallet connect to a trustworthy blockchain node?
- Does the application allow users to configure a custom blockchain node? If so, what can a malicious blockchain node do to the application?
- Does the application utilize a centralized server? What information is sent from the client to the server?
- If the server stores sensitive data, how is it stored?
- Does the application enforce a strong password policy?
- Does the application require 2FA or a PIN code when users attempt to access sensitive information or transfer tokens?
- Does the application use vulnerable third-party libraries?
- Are there any secrets (e.g., API keys, AWS credentials) leaked in the source code repository?
- Are there any notable bad coding practices (e.g., misuse of cryptography) in the codebase?
- Does the application server enforce TLS connections?

**Threat hypothesis related to the browser extension**

- What permissions does the extension require, and are they minimal and necessary?
- How does the extension decide which website is allowed to communicate with it?
- How does the extension interact with the web page?
- Is the extension vulnerable to clickjacking?
- Does the application implement an effective content security policy?
- Can a malicious website read or modify data that belongs to the extension without the user's consent?
- Does the extension (often the background script) correctly check the origin of the message before processing it?
- Can a malicious website exploit vulnerabilities such as XSS in the extension page or other active tabs in the browser by exploiting a vulnerability in the extension?
- Have cross-origin requests and interactions made by the extension been evaluated to ensure adherence to the same-origin policy and proper use of CORS (Cross-Origin Resource Sharing)?
- Has the message-passing mechanism between the extension's content scripts and background scripts been evaluated for integrity and security?
- Does the extension communicate with third-party APIs, and is it secure and does not leak sensitive information?
- Is the extension vulnerable to social engineering attacks, which could trick users into unauthorized transactions or revealing sensitive information?

- Is the extension's UI vulnerable to overlay attacks or manipulations that could deceive users into performing unintended actions?

### Limitations

The following functionality was not tested during the assessment because it was not implemented:

- Settings > Security Audit

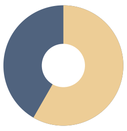
### Out of scope dependency

The extension wallet codebase serves as the underlying entity to interact with one or more out-of-scope dependencies defined in the `package.json` file. The scope of the audit treats out-of-scope dependencies as black boxes and assumes their functional correctness. However, in the real world, dependencies can be compromised, resulting in user experience issues, application operation disruption, or asset loss. The auditors recommend that the team continually monitor the status of out-of-scope packages. Establishing an automated alert system for changes or anomalies can help mitigate potential risks when unexpected activities occur.

### Report Scope

The current report is intended for the client-side application of the Kucoin extension wallet. Server-side APIs are covered in a separate report.

## FINDINGS | KUCOIN EXTENSION WALLET - CLIENT SIDE



| 12 | 0 | 0 | 0 | 7 | 5 |
|---|---|---|---|---|---|
| Total Findings | Critical | High | Medium | Low | Informational |

This report has been prepared for Kucoin to identify potential vulnerabilities and security issues within the reviewed codebase. During the course of the security assessment, a total of 12 issues were identified. Leveraging a combination of Dynamic Testing, Manual Review & Testnet Deployment the following findings were uncovered:

| ID | Title | Category | Severity | Status |
|---|---|---|---|---|
| KEW-01 | Plaintext Session Secret Leads To MasterKey Exposure | Insecure Data Storage | Low | ● Resolved |
| KEW-02 | Locked Wallet Disclose Its Addresses | Logic Flaws | Low | ● Resolved |
| KEW-05 | DApp Blacklist Bypass | Logic Flaws | Low | ● Resolved |
| KEW-11 | DApp Spam Blocking Not Enforced | Logic Flaws | Low | ● Resolved |
| KEW-12 | Permit Signing Shows Raw Timestamp Instead Of Expiry Date | Logic Flaws | Low | ● Resolved |
| KEW-16 | Overly Permissive `object-src` In CSP | Security Misconfiguration | Low | ● Resolved |
| KEW-17 | Using Components With Known Vulnerabilities | Security Misconfiguration | Low | ● Resolved |
| KEW-04 | Premature Wallet Unlock During Password Change Flow | Logic Flaws | Informational | ● Resolved |
| KEW-08 | Non-Persistent Device ID In Service Worker | Logic Flaws | Informational | ● Resolved |
| KEW-10 | Usage Of `innerHTML` | Injection | Informational | ● Resolved |
| KEW-15 | No Structured Display For EIP-4361 SIWE Messages | Logic Flaws | Informational | ● Resolved |

| ID | Title | Category | Severity | Status |
|---|---|---|---|---|
| KEW-18 | Unused `cookies` Permission In Manifest | Security Misconfiguration | Informational | ● Resolved |

## KEW-01 | Plaintext Session Secret Leads To MasterKey Exposure

| Category | Severity | Location | Status |
|---|---|---|---|
| Insecure Data Storage | ● Low | web3-wallet-extension/src/background/service/keyring/index.ts web3-wallet-extension/src/background/utils/sessionUnlock.ts | ● Resolved |

### Description

In a secure design, the masterKey should only be decrypted using the user password from `wallet_session_master_vault` . The masterKey is the core key used to decrypt mnemonic and private key seeds. However, during mnemonic creation, the extension stores a second encrypted copy of the masterKey. This copy is encrypted with a random `wallet_session_secret` , and both the encrypted vault( `wallet_session_master_vault` ) and the plaintext secret( `wallet_session_secret` ) are stored together in `chrome.session` storage.

### Impact

If an attacker gains access to `chrome.session` storage, they can obtain both `wallet_session_master_vault` and `wallet_session_secret` . With these two values, the masterKey can be decrypted directly. Additionally, the user password is not required in this process. As a result, the attacker can decrypt seeds and fully compromise mnemonic phrases and private keys.

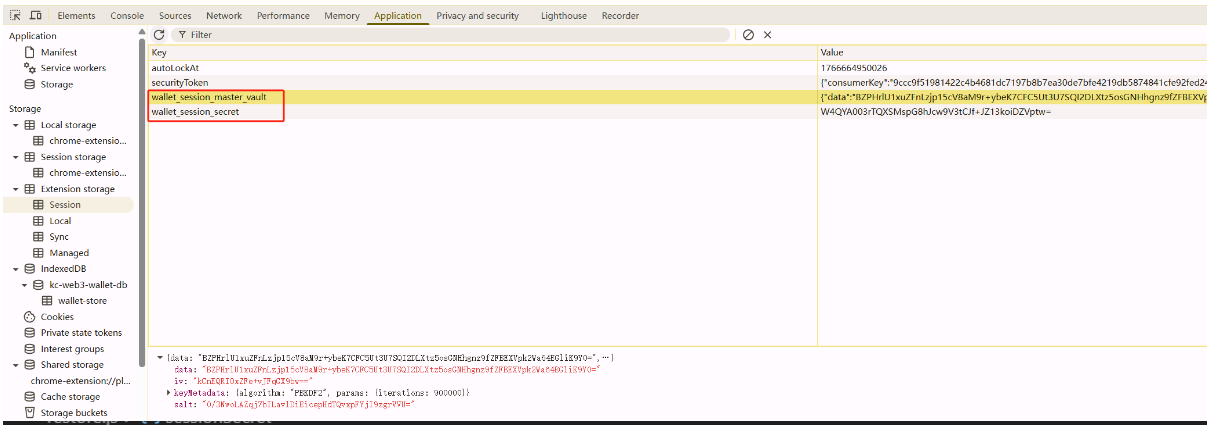
### Proof of Concept

1. The wallet extension stores both the encrypted masterKey( `wallet_session_master_vault` ) and the associated plaintext decryption key( `wallet_session_secret` ) in `chrome.storage.session` , as shown below.

```
//web3-wallet-extension/src/background/service/keyring/index.ts
  async boot(password: string) {
    if (this.isBooted()) throw new Error('is booted');

    // 1. 生成随机 masterKey
    const masterKey = generateMasterKey();

    // 2. 用密码加密 masterKey，得到 masterVault
    const masterVault = await encryptMasterKeyWithPassword(password, masterKey);
    await walletDB.set(WALLET_DB_KEYS.WALLET_MASTER_VAULT, masterVault);
    // await walletDB.set(WALLET_DB_KEYS.WALLET_PASSWORD_TIP, passwordTip);

    // 3. 原来的 booted 标记可以简化成固定字符串（仅逻辑哨兵用）
    const booted = 'true';
    await walletDB.set(WALLET_DB_KEYS.WALLET_BOOTED, booted);

    ...
    // 5. 会话自动解锁：初次设置成功后写入 session，便于 SW 重启时免密恢复
    await setupSessionAutoUnlock(masterKey);
  }
```

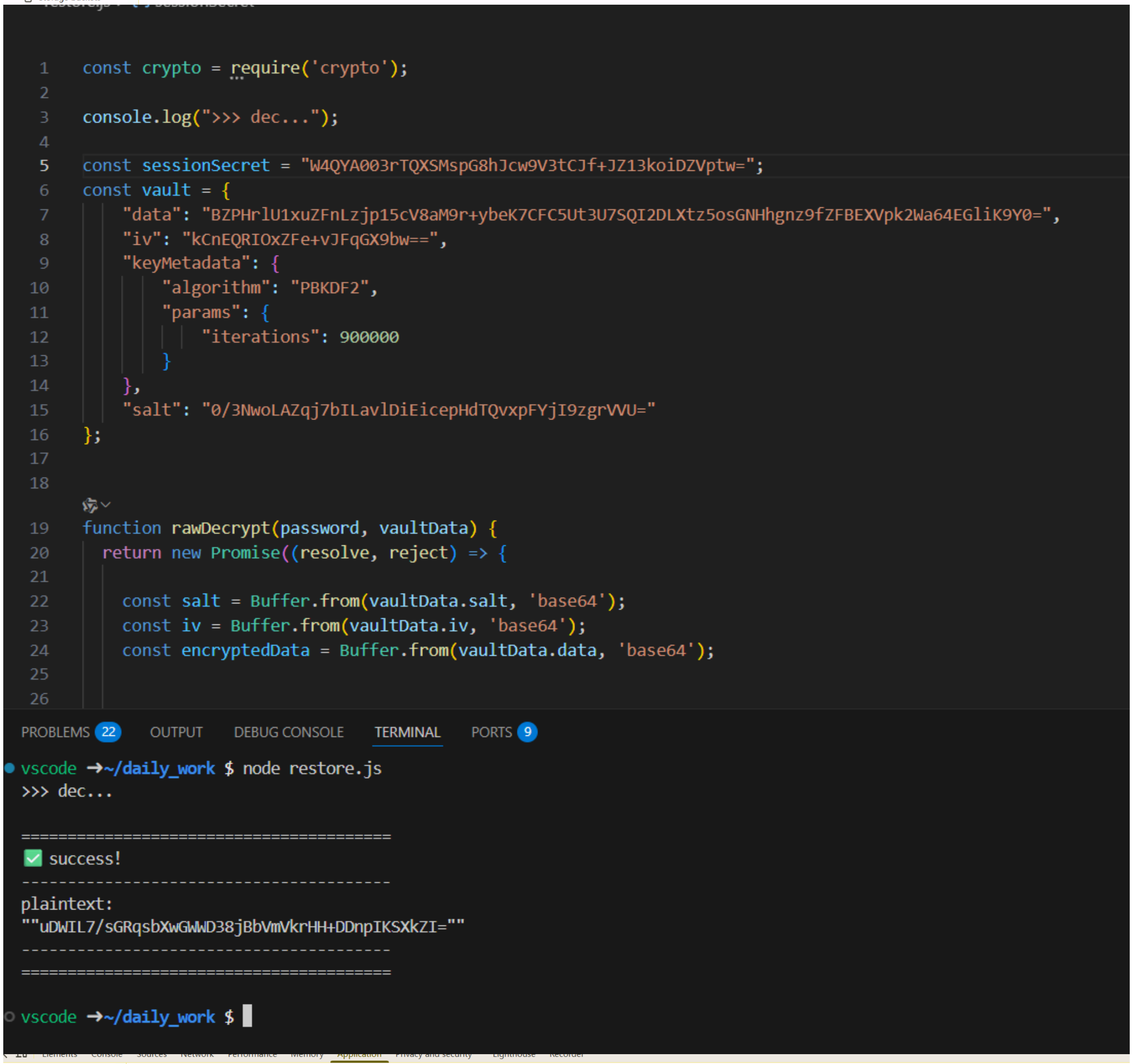
```
//web3-wallet-extension/src/background/utils/sessionUnlock.ts
  export const setupSessionAutoUnlock = async (masterKey: string) => {
  // 1. 先看 session 里有没有 secret，没有就生成一个
  const [sessionSecretFromStore] = await Promise.all([
    browserSession.get(SESSION_SECRET_KEY),
    browserSession.get(SESSION_MASTER_VAULT_KEY),
  ]);

  let sessionSecret = sessionSecretFromStore as string | undefined;
  if (!sessionSecret) {
    sessionSecret = generateSessionSecret();
  }

  // 2. 用 sessionSecret 加密 masterKey（这层只用于“方便型自动解锁”）
  const { vault } = await encryptWithDetail(sessionSecret, masterKey);

  // 3. 两者均写入 sessionStorage
  await Promise.all([
    browserSession.set(SESSION_SECRET_KEY, sessionSecret),
    browserSession.set(SESSION_MASTER_VAULT_KEY, vault),
  ]);
};
```

2. Decrypt the masterKey using the plaintext-stored wallet_session_secret, then decrypt the mnemonic using the masterKey without the user's password



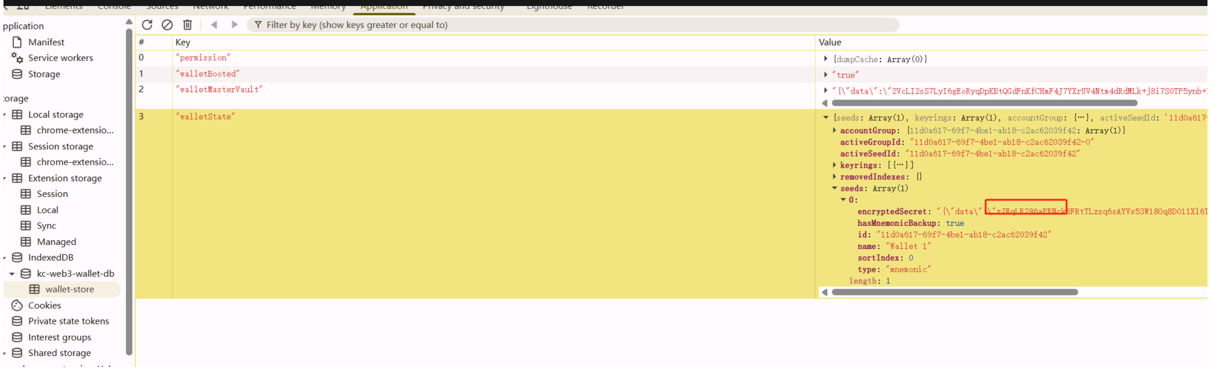
| Key | Value |
|---|---|
| autoLockAt | 1766664950026 |
| securityToken | {"consumerKey":"9ccc9f51981422c4b4681dc7197b8b7ea30de7bfe4219db5874841cfe92fed2 |
| wallet_session_master_vault | {"data":"BZPHrlU1xuZFnLzjp15cV8aM9r+ybeK7CFC5Ut3U7SQI2DLXtz5osGNHhgnz9fZFBEXVp |
| wallet_session_secret | W4QYA003rTQXSMspG8hJcw9V3tCJf+JZ13koiDZVptw= |

```
▾ {data: "BZPHrlU1xuZFnLzjp15cV8aM9r+ybeK7CFC5Ut3U7SQI2DLXtz5osGNHhgnz9fZFBEXVpk2Wa64EGliK9Y0=", …}
    data: "BZPHrlU1xuZFnLzjp15cV8aM9r+ybeK7CFC5Ut3U7SQI2DLXtz5osGNHhgnz9fZFBEXVpk2Wa64EGliK9Y0="
    iv: "kCnEQRIOxZFe+vJFqGX9bw=="
  ▸ keyMetadata: {algorithm: "PBKDF2", params: {iterations: 900000}}
    salt: "0/3NwoLAZqj7bILavlDiEicepHdTQvxpFYjI9zgrVVU="
```



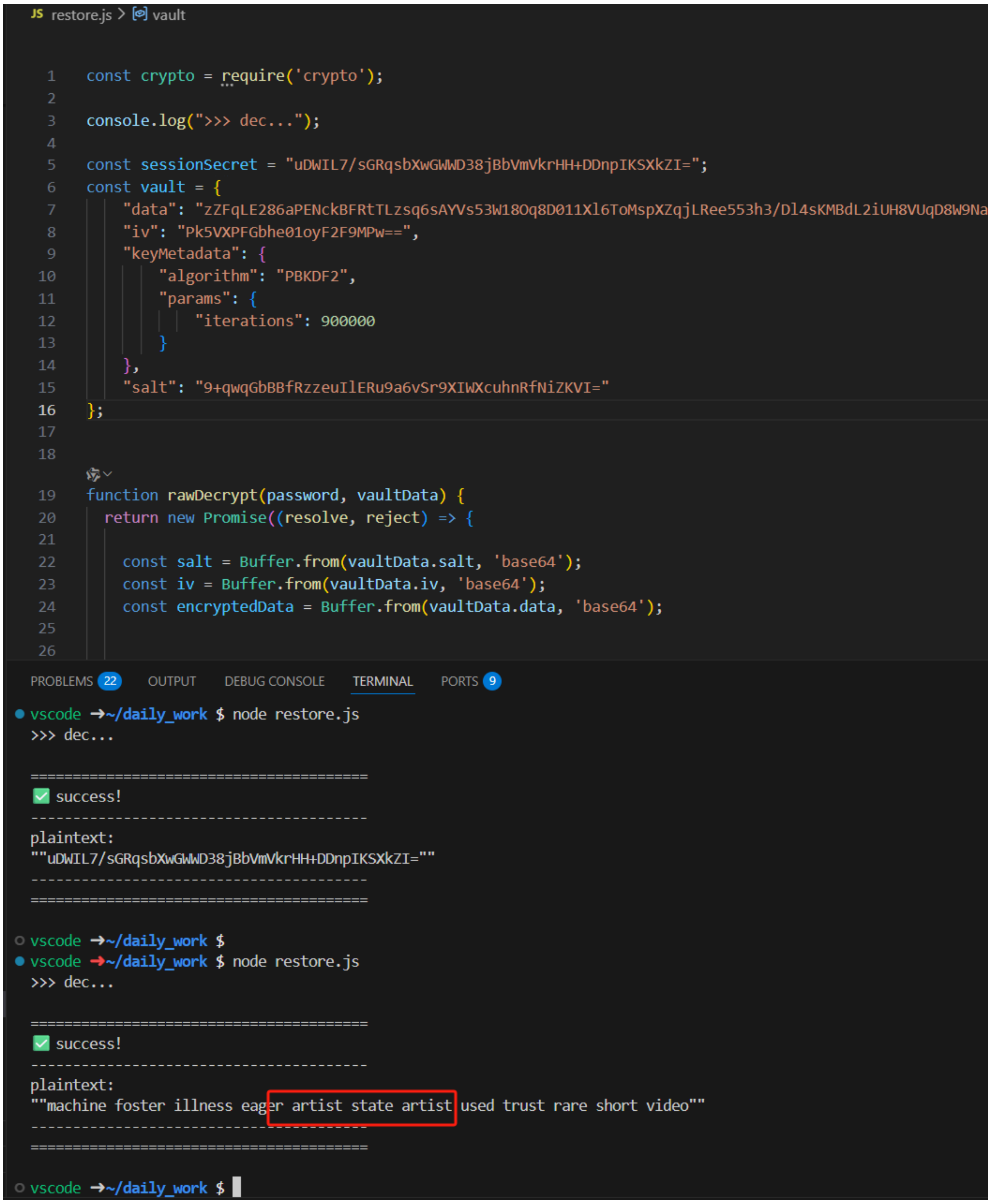
```javascript
const crypto = require('crypto');

console.log(">>> dec...");

const sessionSecret = "W4QYA003rTQXSMspG8hJcw9V3tCJf+JZ13koiDZVptw=";
const vault = {
    "data": "BZPHrlU1xuZFnLzjp15cV8aM9r+ybeK7CFC5Ut3U7SQI2DLXtz5osGNHhgnz9fZFBEXVpk2Wa64EGliK9Y0=",
    "iv": "kCnEQRIOxZFe+vJFqGX9bw==",
    "keyMetadata": {
        "algorithm": "PBKDF2",
        "params": {
            "iterations": 900000
        }
    },
    "salt": "0/3NwoLAZqj7bILavlDiEicepHdTQvxpFYjI9zgrVVU="
};


function rawDecrypt(password, vaultData) {
  return new Promise((resolve, reject) => {

    const salt = Buffer.from(vaultData.salt, 'base64');
    const iv = Buffer.from(vaultData.iv, 'base64');
    const encryptedData = Buffer.from(vaultData.data, 'base64');

```

```
vscode ➜~/daily_work $ node restore.js
>>> dec...

========================================
✅ success!
----------------------------------------
plaintext:
""uDWIL7/sGRqsbXwGWWD38jBbVmVkrHH+DDnpIKSXkZI=""
----------------------------------------
========================================

vscode ➜~/daily_work $ ▌
```



| # | Key | Value |
|---|---|---|
| 0 | "permission" | ▸ {dumpCache: Array(0)} |
| 1 | "walletBooted" | ▸ "true" |
| 2 | "walletMasterVault" | ▸ "{\"data\":\"2VcLI2sS7LyI6gEoRyqDpKBtQGdFnKfCHmF4J7YXrUV4Ntm4dRdMLk+j8i7S0TP5ynb+ |
| 3 | "walletState" | |

```
▾ {seeds: Array(1), keyrings: Array(1), accountGroup: {…}, activeSeedId: '11d0a617-
  ▸ accountGroup: {11d0a617-69f7-4be1-ab18-c2ac62039f42: Array(1)}
    activeGroupId: "11d0a617-69f7-4be1-ab18-c2ac62039f42-0"
    activeSeedId: "11d0a617-69f7-4be1-ab18-c2ac62039f42"
  ▸ keyrings: [{…}]
  ▸ removedIndexes: {}
  ▾ seeds: Array(1)
    ▾ 0:
        encryptedSecret: "{\"data\":\"z7EqLF286aPENck3FRtTLzsq6sAYVs53W18Oq8D011X16T
        hasMnemonicBackup: true
        id: "11d0a617-69f7-4be1-ab18-c2ac62039f42"
        name: "Wallet 1"
        sortIndex: 0
        type: "mnemonic"
      length: 1
```



```
const crypto = require('crypto');

console.log(">>> dec...");

const sessionSecret = "uDWIL7/sGRqsbXwGWWD38jBbVmVkrHH+DDnpIKSXkZI=";
const vault = {
    "data": "zZFqLE286aPENckBFRtTLzsq6sAYVs53W18Oq8D011Xl6ToMspXZqjLRee553h3/Dl4sKMBdL2iUH8VUqD8W9Na
    "iv": "Pk5VXPFGbhe01oyF2F9MPw==",
    "keyMetadata": {
        "algorithm": "PBKDF2",
        "params": {
            "iterations": 900000
        }
    },
    "salt": "9+qwqGbBBfRzzeuIlERu9a6vSr9XIWXcuhnRfNiZKVI="
};


function rawDecrypt(password, vaultData) {
  return new Promise((resolve, reject) => {

    const salt = Buffer.from(vaultData.salt, 'base64');
    const iv = Buffer.from(vaultData.iv, 'base64');
    const encryptedData = Buffer.from(vaultData.data, 'base64');


```

```
● vscode ➜~/daily_work $ node restore.js
  >>> dec...

  ======================================
  ✅ success!
  --------------------------------------
  plaintext:
  ""uDWIL7/sGRqsbXwGWWD38jBbVmVkrHH+DDnpIKSXkZI=""
  --------------------------------------
  ======================================

○ vscode ➜~/daily_work $
● vscode ➜~/daily_work $ node restore.js
  >>> dec...

  ======================================
  ✅ success!
  --------------------------------------
  plaintext:
  ""machine foster illness eager artist state artist used trust rare short video""
  --------------------------------------
  ======================================

○ vscode ➜~/daily_work $ ▌
```

## Recommendation

It is recommended that the session secret used to encrypt the masterKey ( `wallet_session_secret` ) is never stored in plaintext alongside its corresponding encrypted data in any storage, including `chrome.session` . Redesign the session unlocking mechanism to ensure the `masterKey` remains accessible only via user password authentication. Session secrets, if necessary, should be derived or protected in a way that their compromise does not enable decryption of critical secrets without additional authentication.

## Alleviation

**[Kucoin, 01/06/2026]**: The team heeded the advice and resolved the issue.

## KEW-02 | Locked Wallet Disclose Its Addresses

| Category | Severity | Location | Status |
|---|---|---|---|
| Logic Flaws | ● Low | | ● Resolved |

### Description

The `initialize()` function retrieves `{ chainId, accounts, networkVersion, isUnlocked }` using the `getProviderState` method. While it updates `_isUnlocked` and `_state.isUnlocked` only when `isUnlocked` is true, it always propagates the accounts array to the public provider state and triggers related events, regardless of the lock status. Specifically, `initialize()` always calls `this._pushEventHandlers.accountsChanged(accounts)`, and `accountsChanged(accounts)` then, updates and emits account state.

```
initialize = async () => {
    // ...
    try {
      // 它负责从 插件 background 脚本 里获取当前钱包的内部状态
      const { chainId, accounts, networkVersion, isUnlocked }: any =
        await this.requestInternalMethods({
          method: 'getProviderState',
        });
      if (isUnlocked) {
        this._isUnlocked = true;
        this._state.isUnlocked = true;
      }
      this.chainId = chainId;
      this.networkVersion = networkVersion;
      this.emit('connect', { chainId });
      this._pushEventHandlers.chainChanged({
        chain: chainId,
        networkVersion,
      });
      this._pushEventHandlers.accountsChanged(accounts);
    } catch (error) {
    } finally {
      this._initialized = true;
      this._state.initialized = true;
      this.emit('_initialized');
    }
  };
```

### Impact

Connected DApps can retrieve the user's wallet address even when the wallet is locked, enabling passive tracking, cross-site correlation, and loss of privacy without explicit user interaction. This undermines expected lock-state privacy and may conflict with consent expectations around account exposure.

## Proof of Concept

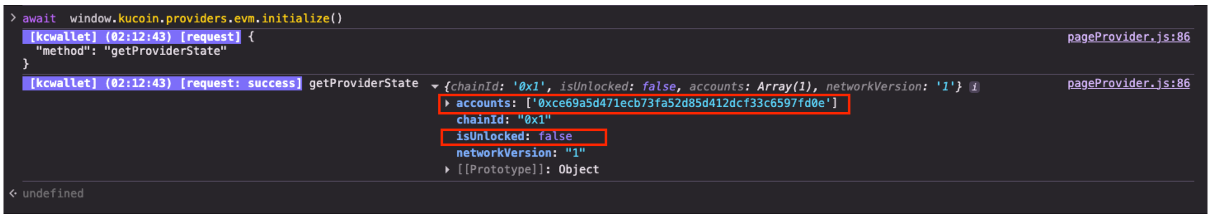
1. Open a webpage that can access the injected provider (e.g., `window.kucoin`).
2. Before calling anything, register an accounts listener to capture the leak:

```
window.kucoin.on('accountsChanged', (accounts) => {
  console.log('leaked account:', accounts[0]);
});
```

3. Trigger initialization explicitly (works even if it already ran once; the leak happens whenever it runs and receives accounts):

```
await window.kucoin.providers.evm.initialize()
```

4. Ensure the wallet is in a locked state. Despite `isUnlocked === false`, `initialize()` will still call `accountsChanged(accounts)` and the listener from step 2 will receive the address.



```
> await  window.kucoin.providers.evm.initialize()
  [kcwallet] (02:12:43) [request] {                                                              pageProvider.js:86
    "method": "getProviderState"
  }
  [kcwallet] (02:12:43) [request: success] getProviderState  ▾ {chainId: '0x1', isUnlocked: false, accounts: Array(1), networkVersion: '1'}   pageProvider.js:86
                                                               ▸ accounts: ['0xce69a5d471ecb73fa52d85d412dcf33c6597fd0e']
                                                                 chainId: "0x1"
                                                                 isUnlocked: false
                                                                 networkVersion: "1"
                                                               ▸ [[Prototype]]: Object
<- undefined
```

5. The `getProviderState` function can be called directly to obtain the same information.

```
await kucoin.providers.evm.requestInternalMethods({ method: 'getProviderState',
params: [] })
```

## Recommendation

The audit team recommends that all logic responsible for updating the public provider state or emitting account-related events—including account addresses—should explicitly check that the wallet is unlocked before proceeding. Account information must not be exposed to DApps, external listeners, or the provider state while the wallet remains locked. Update the initialization and event-handling flow so that account data is only propagated when the wallet has been verified as unlocked, protecting user privacy as expected.

## Alleviation

**[Kucoin, 01/07/2026]**: The team heeded the advice and resolved the issue by preventing account data from being returned while the wallet is locked, eliminating unintended address disclosure.

## KEW-05 | DApp Blacklist Bypass

| Category | Severity | Location | Status |
|---|---|---|---|
| Logic Flaws | ● Low | web3-wallet-extension/src/ui/views/Approval/index.tsx:81 web3-wallet-extension/src/utils/index.ts:173 | ● Resolved |

### Description

The approval flow normalizes the requesting origin using `stripProtocol(origin)` and then checks membership against local allow/deny lists:

```
// src/ui/views/Approval/index.tsx
const normalizeOrigin = stripProtocol(origin);
if (dappStore.dappWhitelist.includes(normalizeOrigin)) { ... }
else if (dappStore.dappBlacklist.includes(normalizeOrigin)) { ... }
else { dappSecurityCheckHandle(normalizeOrigin); }
```

However, `stripProtocol` returns `URL.host` instead of `URL.hostname` :

```
// src/utils/index.ts
export function stripProtocol(url: string = ''): string {
  const u = new URL(url);
  return u.host; // includes port (e.g., "evil.com:8443")
}
```

Because `host` includes the port, entries stored as plain hostnames (e.g., `evil.com` ) will not match when the same site is accessed with an explicit port (e.g., `https://evil.com:8443` ). This causes the local blacklist check to miss and fall through to alternative handling.

### Impact

A blacklisted DApp can evade local blocking by serving content on a non-default port, allowing approval flows to proceed when they should be denied.

### Recommendation

It is recommended to normalize origins using the hostname only (excluding ports) before performing allow/deny checks, and apply a single canonical normalization strategy across UI and backend.

### Alleviation

**[Kucoin, 01/07/2026]**: The team heeded the advice and resolved the issue by normalizing origins using `URL.hostnam` e instead of `URL.host`, ensuring blacklist and whitelist checks are not bypassed via explicit ports.

## KEW-11 | DApp Spam Blocking Not Enforced

| Category | Severity | Location | Status |
|---|---|---|---|
| Logic Flaws | ● Low | web3-wallet-extension/src/background/service/notification.ts | ● Resolved |

### Description

The `requestApproval` flow includes logic intended to temporarily block a DApp for one minute by checking `dappManager.get(origin)?.isBlocked` and `blockedTimestamp`. If a DApp is marked as blocked and the timestamp is recent, the request is immediately rejected to prevent repeated approval popups. However, the supporting rejection-tracking logic (`addLastRejectDapp` and related code paths) does not actually set `isBlocked = true` or update `blockedTimestamp` based on rejection counters (e.g., `lastRejectCount`). As a result, the guard in `requestApproval` never triggers in practice and requests are not throttled.

```
// web3-wallet-extension/src/background/service/notification.ts
requestApproval = async (data, winProps?): Promise<any> => {
    const origin = this.getOrigin(data);
    if (origin) {
      const dapp = this.dappManager.get(origin);
      if (dapp?.isBlocked && Date.now() - dapp.blockedTimestamp < 60 * 1000 * 1) {
        throw ethErrors.provider.userRejectedRequest('User rejected the request.');
      }
    }

// ...

 private addLastRejectDapp() {
    if (this.currentApproval?.data?.params?.$ctx) return;
    const origin = this.getOrigin();
    if (!origin) {
      return;
    }
    const dapp = this.dappManager.get(origin);
    // same origin and less 1 min
    if (dapp && Date.now() - dapp.lastRejectTimestamp < 60 * 1000) {
      dapp.lastRejectCount = dapp.lastRejectCount + 1;
      dapp.lastRejectTimestamp = Date.now();
    } else {
      this.dappManager.set(origin, {
        lastRejectTimestamp: Date.now(),
        lastRejectCount: 1,
        blockedTimestamp: 0,
        isBlocked: false,
      });
    }
  }
```

## Impact

Users can be overwhelmed with repeated prompts, making the extension difficult to use and increasing the likelihood of accidental approvals.

## Recommendation

It is recommended to properly set the DApp as blocked when the rejection threshold is reached. Ensure that after a configurable number of rapid rejections (using `lastRejectCount` within a specified time window), `isBlocked` is set to true and `blockedTimestamp` is updated accordingly.

## Alleviation

**[Kucoin, 01/07/2026]**: The team heeded the advice and resolved the issue by blocking DApps after repeated rapid rejections and enforcing a temporary cooldown.

## KEW-12 | Permit Signing Shows Raw Timestamp Instead Of Expiry Date

| Category | Severity | Location | Status |
|---|---|---|---|
| Logic Flaws | ● Low | projects/web3-wallet-extension/src/ui/views/Approval/components/SignTypedData/index.tsx | ● Resolved |

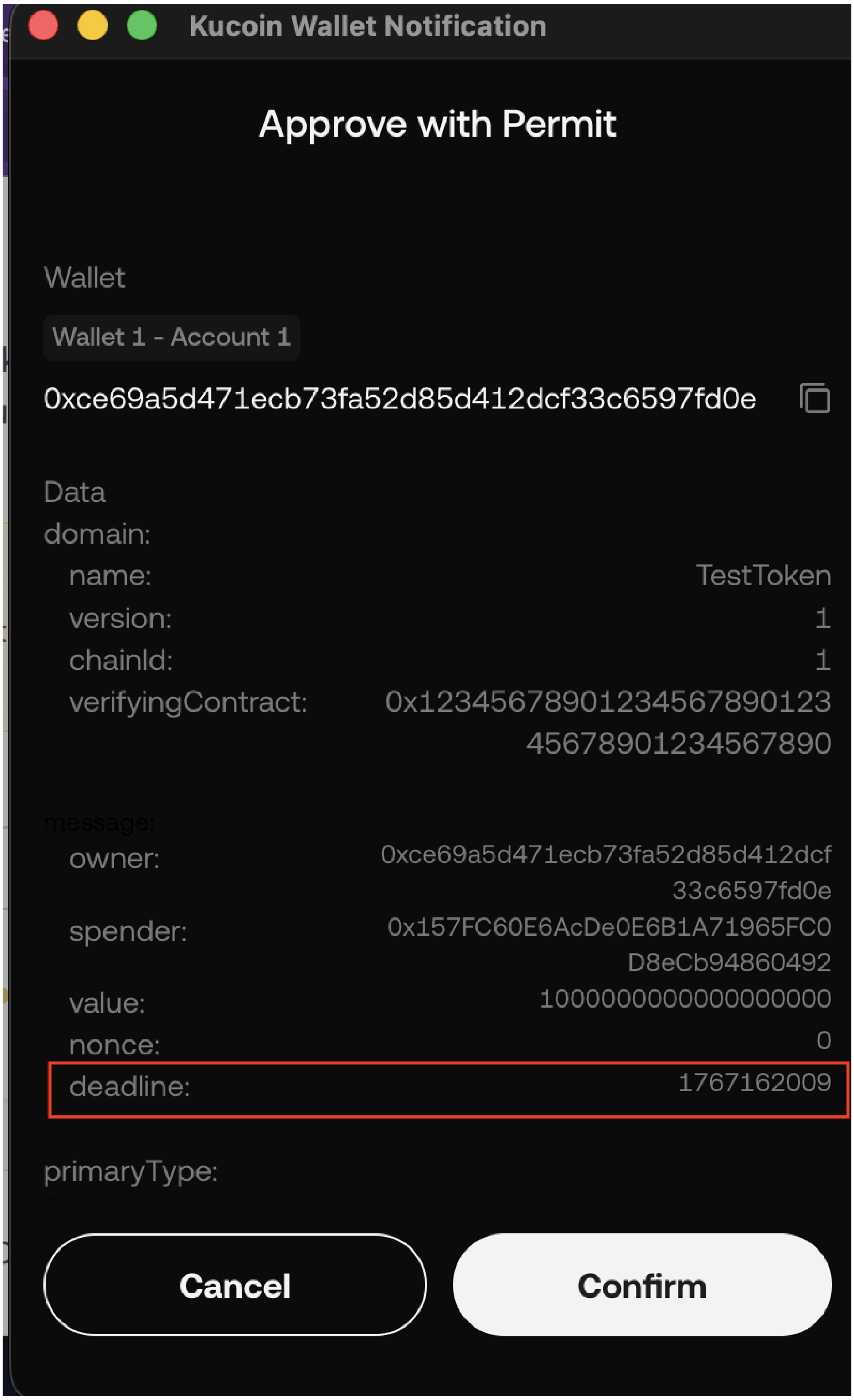
### Description

When a DApp prompts the user to sign a Permit request, the wallet displays the expiry as a Unix timestamp rather than a user-friendly date and time. Presenting the expiration in this format makes it difficult for users to understand the duration of the approval period.

### Impact

Users may misunderstand the duration of the authorization and approve permits that last longer than intended, or reject legitimate requests due to confusion.

### Proof of Concept



Approve with Permit

Wallet

Wallet 1 - Account 1

0xce69a5d471ecb73fa52d85d412dcf33c6597fd0e

Data

domain:

| | |
|---|---|
| name: | TestToken |
| version: | 1 |
| chainId: | 1 |
| verifyingContract: | 0x1234567890123456789012345678901234567890 |

message:

| | |
|---|---|
| owner: | 0xce69a5d471ecb73fa52d85d412dcf33c6597fd0e |
| spender: | 0x157FC60E6AcDe0E6B1A71965FC0D8eCb94860492 |
| value: | 1000000000000000000 |
| nonce: | 0 |
| deadline: | 1767162009 |

primaryType:

## Recommendation

It is recommended to convert the raw Unix timestamp to a clear, human-readable date and time format. This will help users better understand the validity period of their approval and make informed decisions when interacting with Permit requests.

## Alleviation

**[Kucoin, 01/07/2026]**: The team heeded the advice and resolved the issue.

## KEW-16 | Overly Permissive `object-src` In CSP

| Category | Severity | Location | Status |
|---|---|---|---|
| Security Misconfiguration | ● Low | web3-wallet-extension/src/manifest.ts | ● Resolved |

### Description

The wallets' Content Security Policy sets `object-src 'self'`, which allows the application to load plugin-backed embedded content (e.g., `<object>`, `<embed>`, `<applet>`) from the same origin. In modern web applications, these elements are typically unnecessary and represent a legacy execution surface that is commonly restricted.

```
content_security_policy: {
    extension_pages: "script-src 'self' 'wasm-unsafe-eval'; object-src 'self'",
  },
```

### Impact

If an attacker finds a way to inject HTML or control same-origin content, allowing `object-src 'self'` can broaden the available vectors to deliver active embedded content, increasing the likelihood of exploitation paths that are otherwise blocked when plugins/embedded objects are disabled. Even when no immediate exploit is present, this setting keeps an avoidable attack surface enabled.

### Recommendation

It is recommended to set `object-src 'none'` unless the application has a strict, validated requirement for embedded object content. If embedded objects are required, scope the directive as narrowly as possible and ensure the application cannot be coerced into loading attacker-controlled same-origin resources through upload, redirect, or injection paths.

### Alleviation

**[Kucoin, 01/07/2026]**: The team heeded the advice and resolved the issue by setting `object-src 'none'` in the Content Security Policy.

## KEW-17 | Using Components With Known Vulnerabilities

| Category | Severity | Location | Status |
|---|---|---|---|
| Security Misconfiguration | ● Low | | ● Resolved |

### Description

Web applications typically rely on several open-source components. Developers can develop rich and complex applications with little cost and effort by integrating open-source development tools, frameworks, and languages. Security researchers and testers typically use automated tools to uncover compromised or vulnerable components, then publish their findings on issue trackers, security advisories, or the National Vulnerability Database (NVD). Any competent attacker who finds this information can use it to exploit particular application surfaces.

The result of the "npm audit" command shows the application is using outdated libraries with known vulnerabilities.

### Impact

Using unmaintained or outdated dependencies may lead to critical vulnerabilities in the code base. An attacker can use publicly known vulnerabilities in outdated libraries to gain unauthorized access, manipulate application data, or disrupt service availability. These actions can lead to data breaches, reputation damage, and potential financial loss.

### Proof of Concept

- web3-wallet-page-provider

```
npm audit --omit dev
# npm audit report

nanoid  <3.3.8
Severity: moderate
Predictable results in nanoid generation when given non-integer values -
https://github.com/advisories/GHSA-mwcw-c2x4-8c55
fix available via `npm audit fix --force`
Will install nanoid@3.3.11, which is outside the stated dependency range
node_modules/nanoid

1 moderate severity vulnerability
```

- web3-wallet-extension

```
# npm audit report

nanoid  <3.3.8
Severity: moderate
Predictable results in nanoid generation when given non-integer values -
https://github.com/advisories/GHSA-mwcw-c2x4-8c55
fix available via `npm audit fix --force`
Will install nanoid@3.3.11, which is outside the stated dependency range
node_modules/nanoid

qs  <6.14.1
Severity: high
qs's arrayLimit bypass in its bracket notation allows DoS via memory exhaustion -
https://github.com/advisories/GHSA-6rw7-vpxm-498p
fix available via `npm audit fix --force`
Will install qs@6.14.1, which is outside the stated dependency range
node_modules/qs

2 vulnerabilities (1 moderate, 1 high)
```

## Recommendation

There should be a patch management process in place to:

- Remove unused dependencies, unnecessary features, components, files, and documentation.
- Continuously inventory the versions of both client-side and server-side components (e.g., frameworks, libraries) and their dependencies using tools like versions, DependencyCheck, retire.js, etc. Continuously monitor sources like CVE and NVD for vulnerabilities in the components. Use software composition analysis tools to automate the process. Subscribe to email alerts for security vulnerabilities related to components you use.
- Only obtain components from official sources over secure links. Prefer signed packages to reduce the chance of including a modified, malicious component.
- Monitor for libraries and components that are unmaintained or do not create security patches for older versions. If patching is not possible, consider deploying a virtual patch to monitor, detect, or protect against the discovered issue.

Organizations should ensure that there is an ongoing plan for monitoring, triaging, and applying updates or configuration changes for the lifetime of the application or portfolio.

## Alleviation

**[Kucoin, 01/13/2026]**: The team heeded the advice and resolved the issue by updating all the dependencies.

## KEW-04 | Premature Wallet Unlock During Password Change Flow

| Category | Severity | Location | Status |
|---|---|---|---|
| Logic Flaws | ● Informational | | ● Resolved |

### Description

In the change password page, the UI calls `wallet.unlock(oldPassword)` when the old password input loses focus. This unlocks the wallet before the user confirms the password change and before the full validation process completes. The unlock operation is only used to check whether the old password is correct and is not required at this stage. As a result, the wallet remains unlocked longer than necessary.

### Impact

This issue does not directly allow password bypass or fund theft. However, it unnecessarily extends the unlocked state of the wallet. If future privileged APIs rely on the isUnlocked state, or if race conditions occur, this behavior may increase the attack surface. It represents a violation of the principle of least privilege.

### Proof of Concept

1. Unnecessarily extend the lifecycle of the `masterKey` and `isUnlocked` state

```
//web3-wallet-extension/src/ui/views/ChangePassword/index.tsx
  const onOldPasswordBlur = async () => {
    if (!oldPassword) return;
    try {
      const unlockStatus = await wallet.unlock(oldPassword);
      if (!unlockStatus) {
        setShowOldPasswordError(true);
      } else {
        setShowOldPasswordError(false);
      }
    } catch (error) {}
  };
```

```
//web3-wallet-extension/src/background/service/keyring/index.ts
  async unlock(password?: string): Promise<boolean> {
    try {
      ...
      // 分支 2：有密码 → 标准解锁流程
      const masterVault = await walletDB.get(WALLET_DB_KEYS.WALLET_MASTER_VAULT);
      if (!masterVault) throw new Error('No masterVault found');

      const masterKey = await decryptMasterKeyWithPassword(password, masterVault as
string);

      this.masterKey = masterKey;
      this.isUnlocked = true;


      ...

      return true;
    } catch (err) {
      return false;
    }
  }
```

2. A more appropriate verifyPassword function

```
//web3-wallet-extension/src/background/service/keyring/index.ts
  async verifyPassword(password: string): Promise<boolean> {
    const masterVault = await walletDB.get(WALLET_DB_KEYS.WALLET_MASTER_VAULT);
    if (!masterVault) throw new Error('No masterVault found');

    try {
      await decryptMasterKeyWithPassword(password, masterVault as string);
      return true;
    } catch {
      return false;
    }
  }
```

## Recommendation

It is recommended to decouple password validation from the unlocking process by using a dedicated function like `wallet.verifyPassword` that does not trigger a full unlock. This approach prevents premature exposure and avoids unnecessarily extending the lifecycle of the masterKey and isUnlocked state. The wallet should remain locked throughout the validation phase and only be unlocked during final confirmation. Additionally, implement logic to immediately clear sensitive data and ensure the wallet stays locked if the flow is interrupted or abandoned.

## Alleviation

**[Kucoin, 01/06/2026]**: The team heeded the advice and resolved the issue by using the verifyPassword function.

## KEW-08 | Non-Persistent Device ID In Service Worker

| Category | Severity | Location | Status |
|---|---|---|---|
| Logic Flaws | ● Informational | | ● Resolved |

### Description

The `getUserUniqueId()` function is designed to persist a device identifier using `window.localStorage`. However, in a Manifest V3 extension environment, background service workers do not provide access to `window` or `localStorage`. As a result, `src/utils/storage.ts` detects this and disables localStorage support, causing `getItem` and `setItem` methods to return null.

```
export const getUserUniqueId = () => {
  if (!storage.getItem(BROWSER_EXTENSION_UID_KEY)) {
    const newId = uuid();
    storage.setItem(BROWSER_EXTENSION_UID_KEY, newId);
    return newId;
  }
  return storage.getItem(BROWSER_EXTENSION_UID_KEY);
}
```

Consequently, `getUserUniqueId()` generates a new UUID on each invocation in the service worker and never persists it, while UI/extension pages may persist a different value. This leads to inconsistent `x-device-no` values across contexts and requests.

References: https://developer.chrome.com/docs/extensions/reference/api/storage#can_extensions_use_web_storage_apis

### Impact

The extension produces mismatched or rotating device identifiers, breaking the assumption of a stable “global unique user identifier.” This can cause inconsistent headers (e.g., x-device-no), disrupt backend session or device-binding logic, complicate troubleshooting, and weaken any controls that depend on a consistent device identity.

### Recommendation

It is recommended to use the Chrome Extension Storage API (such as `chrome.storage.local` or `chrome.storage.sync`) to persist the device identifier in all relevant extension contexts, including service workers under Manifest V3. Update the logic in `getUserUniqueId()` to retrieve and store the unique identifier using the Chrome Extension Storage API, ensuring a consistent and persistent device ID is accessible wherever required.

### Alleviation

**[Kucoin, 01/12/2026]**: The team heeded the advice and resolved the issue by persisting the device ID in `browser.storage.local` .

## KEW-10 | Usage Of `innerHTML`

| Category | Severity | Location | Status |
|---|---|---|---|
| Injection | ● Informational | web3-wallet-page-provider/src/chain/evm/notice.ts:51 | ● Resolved |

### Description

The `insert()` function in the `Notice` class calls the insecure function `innerHTML`.

```
// web3-wallet-page-provider/src/chain/evm/notice.ts:51
 insert() {
    if (!this.el) {
      return;
    }

    // main
    const elMain = document.createElement('div');
    elMain.className = 'kucoin-notice-content';
    elMain.innerHTML = this.options.content;
    this.el?.appendChild(elMain);
```

The `notice` function in the same file at line 179 returns a `Notice` object. This function is called from the following files.

- `web3-wallet-page-provider/src/chain/evm/interceptors/switchChain.ts`
- `web3-wallet-page-provider/src/chain/evm/interceptors/switchWallet.ts`

Although the calls are commented out, the intended pattern is considered insecure.

### Impact

In general, any use of `innerHTML` with dynamic strings can create a Cross-Site Scripting (XSS) vulnerability.

### Recommendation

It is recommended to avoid using `innerHTML`. If the application does need to use it, ensure that users cannot control the value passed into the function.

### Alleviation

**[Kucoin, 01/07/2026]**: The team heeded the advice and resolved the issue by remove the notice.ts

## KEW-15 | No Structured Display For EIP-4361 SIWE Messages

| Category | Severity | Location | Status |
|---|---|---|---|
| Logic Flaws | ● Informational | | ● Resolved |

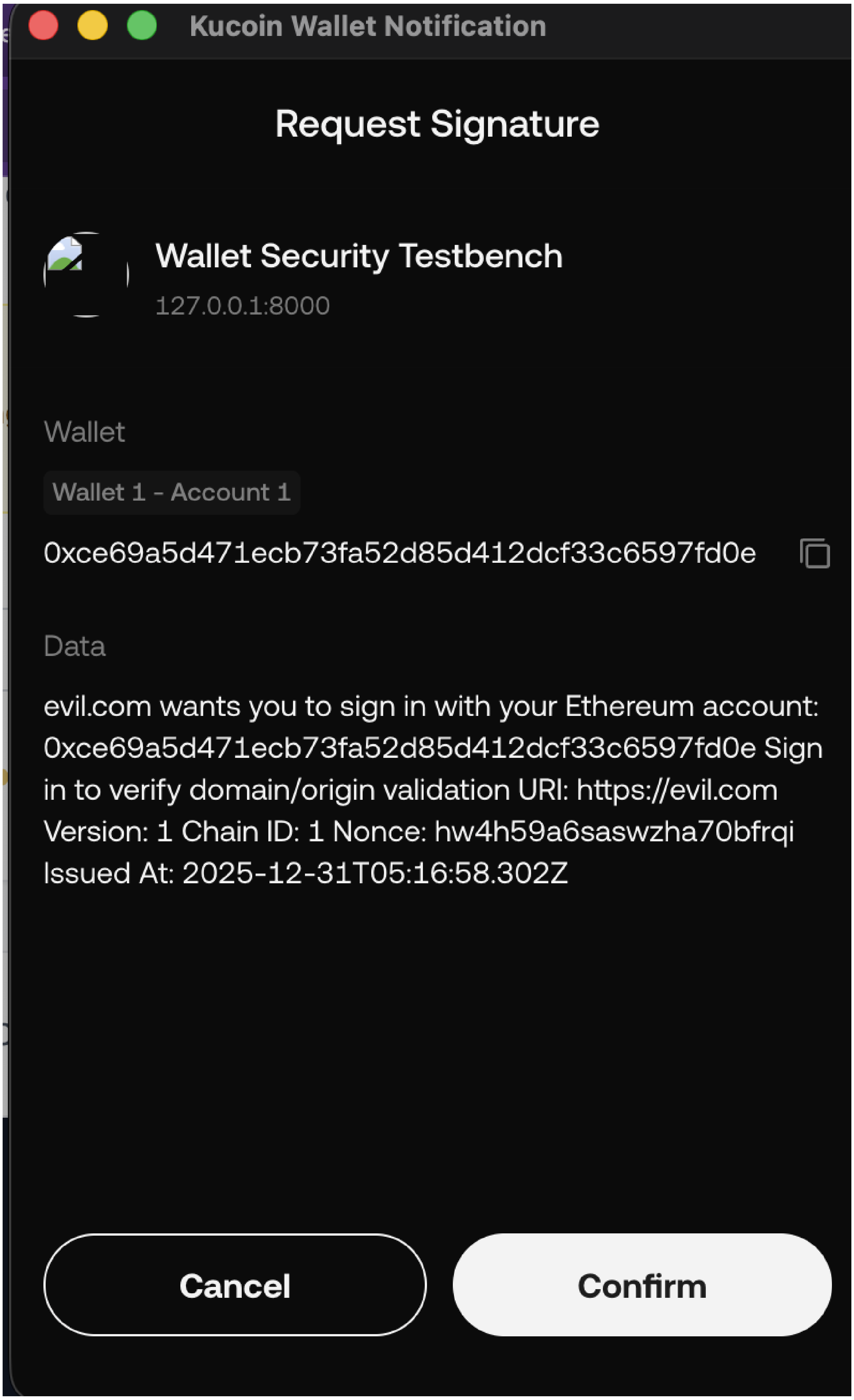
### Description

The wallet currently displays raw EIP-4361 "Sign-In with Ethereum" (SIWE) messages without parsing or presenting them in a structured interface. As a result, users see the full message text instead of a user interface that extracts and emphasizes key SIWE fields, which can make important information less visible and reduce clarity during the sign-in process.

### Impact

Users may have difficulty understanding what they are signing, increasing the likelihood of approving deceptive login prompts.

### Proof of Concept



Request Signature

Wallet Security Testbench
127.0.0.1:8000

Wallet
Wallet 1 - Account 1
0xce69a5d471ecb73fa52d85d412dcf33c6597fd0e

Data
evil.com wants you to sign in with your Ethereum account: 0xce69a5d471ecb73fa52d85d412dcf33c6597fd0e Sign in to verify domain/origin validation URI: https://evil.com Version: 1 Chain ID: 1 Nonce: hw4h59a6saswzha70bfrqi Issued At: 2025-12-31T05:16:58.302Z

## Recommendation

It is recommended to implement a structured, user-friendly display for EIP-4361 SIWE messages in the wallet interface. The display should parse the message and clearly highlight essential fields such as domain, statement, address, nonce, and expiration time, making the sign-in details easy for users to review and understand before approving.

## Alleviation

**[Kucoin, 01/07/2026]**: The team heeded the advice and resolved the issue.

## KEW-18 | Unused `cookies` Permission In Manifest

| Category | Severity | Location | Status |
|---|---|---|---|
| Security Misconfiguration | ● Informational | web3-wallet-extension/src/manifest.ts | ● Resolved |

### Description

The extension requests the `cookies` permission and broad host permissions in its manifest:

```
// src/manifest.ts
permissions: [
  'notifications',
  'storage',
  'cookies',
],
host_permissions: ['<all_urls>'],
```

However, no active use of the extension cookies API ( `chrome.cookies` / `browser.cookies` ) was identified in the codebase. The only cookie-related reference found is commented-out UI theme persistence code using `js-cookie` , which does not rely on the extension-level `cookies` permission:

```
// web3-wallet-extension/src/ui/store/theme.ts (commented)
// import Cookies from 'js-cookie';
// Cookies.set(LS_THEME_KEY, theme, { domain: getCookieDomain(), expires: 365 });
```

This suggests the `cookies` permission is currently unnecessary for the implemented functionality.

### Impact

Requesting unnecessary privileges increases the extension's attack surface and the impact of any extension-context compromise. If the extension is exploited, the `cookies` permission combined with broad host access can enable access to site cookies and session material.

### Recommendation

It is recommended to remove the `cookies` permission unless it is strictly required for a verified feature. Maintain least-privilege by requesting only the permissions needed at runtime.

### Alleviation

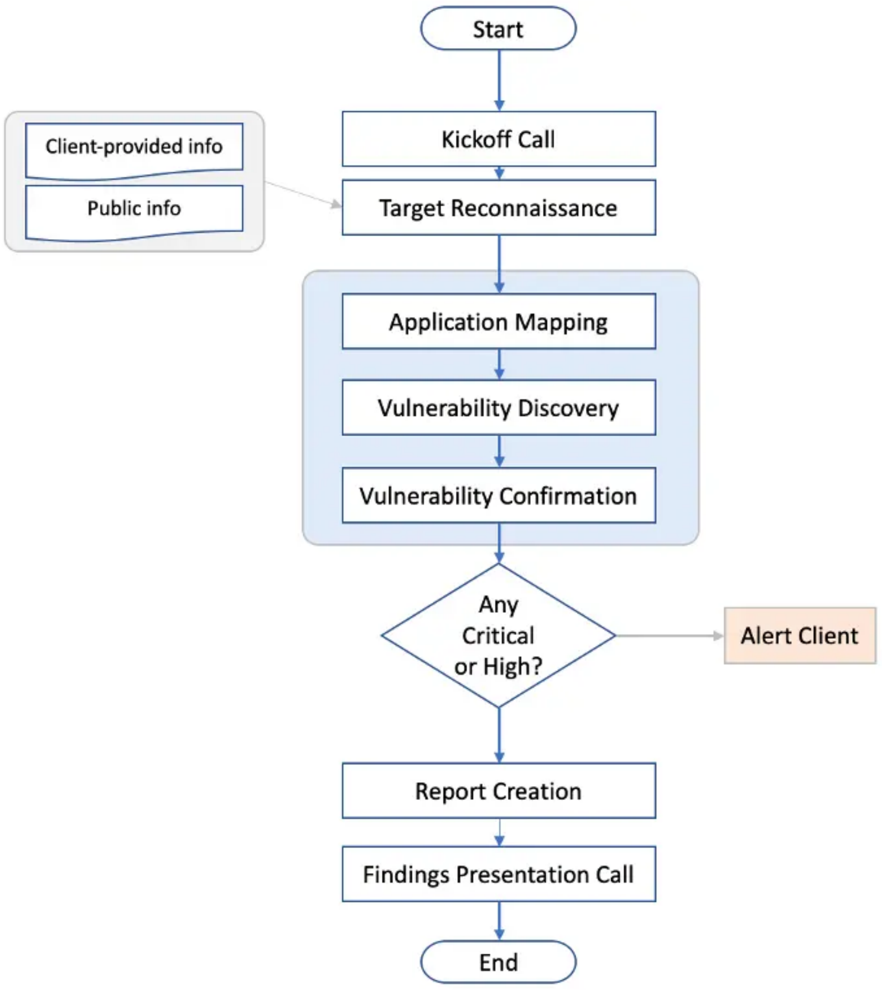
**[Kucoin, 01/07/2026]**: The team heeded the advice and resolved the issue by removing the unnecessary cookies permission from the extension manifest.

# APPENDIX | KUCOIN EXTENSION WALLET - CLIENT SIDE

## Methodology

CertiK uses a comprehensive penetration testing methodology which adheres to industry best practices and standards in security assessments including from OWASP (Open Web Application Security Project), NIST, PTES (Penetration Testing Execution Standard).

Below is a flowchart of our assessment process:



## Coverage and Prioritization

As many components as possible will be tested manually. Priority is generally based on three factors: critical security controls, sensitive data, and the likelihood of vulnerability.

Critical security controls will always receive the top priority in the test. If a vulnerability is discovered in the critical security control, the entire application is likely to be compromised, resulting in a critical-risk to the business. For most applications, critical controls will include the login page, but it could also include major workflows such as the checkout function in an online store.

The Second priority is given to application components that handle sensitive data. This is dependent on business priorities, but common examples include payment card data, financial data, or authentication credentials.

Final priority includes areas of the application that are most likely to be vulnerable. This is based on CertiK' experience with similar applications developed using the same technology or with other applications that fit the same business role. For example, large applications will often have older sections that are less likely to utilize modern security techniques.

## Reconnaissance

CertiK gathers information about the target application from various sources depending on the type of test being performed. CertiK obtains whatever information that is possible and appropriate from the client during scoping and supplements it with relevant information that can be gathered from public sources. This helps provide a better overall picture and understanding of the target.

## Application Mapping

CertiK examines the application, reviewing its contents, and mapping out all its functionalities and components. CertiK makes use of different tools and techniques to traverse the entire application and document all input areas and processes. Automated tools are used to scan the application and it is then manually examined for all its parameters and functionalities. With this, CertiK creates and widens the overall attack surface of the target application.

## Vulnerability Discovery

Using the information that is gathered, CertiK comes up with various attack vectors to test against the application. CertiK uses a combination of automated tools and manual techniques to identify vulnerabilities and weaknesses. Industry-recognized testing tools will be used, including Burp Suite, Nikto, Metasploit, and Kali. Furthermore, any controls in place that would inhibit the successful exploitation of a particular system will be noted.

## Vulnerability Confirmation

After discovering vulnerabilities in the application, CertiK validates the vulnerabilities and assesses its overall impact. To validate, CertiK performs a Proof-of-Concept of an attack on the vulnerability, simulating real world scenarios to prove the risk and overall impact of the vulnerability.

Through CertiK's knowledge and experience on attacks and exploitation techniques, CertiK is able to process all weaknesses and examine how they can be combined to compromise the application. CertiK may use different attack chains, leveraging different weaknesses to escalate and gain a more significant compromise.

To minimize any potential negative impact, vulnerability exploitation was only attempted when it would not adversely affect production applications and systems, and then only to confirm the presence of a specific vulnerability. Any attack with the potential to cause system downtime or seriously impact business continuity was not performed. Vulnerabilities were never exploited to delete or modify data; only read-level access was attempted. If it appeared possible to modify data, this was noted in the list of vulnerabilities below.

## Immediate Escalation of High or Critical Findings

If critical or high findings are found whereby application elements are compromised, client's key security contacts will be notified immediately.

## Risk Assessment

| Risk Level | CVSS Score | Impact | Exploitability |
|---|---|---|---|
| Critical | 9.0-10.0 | Root-level or full-system compromise, large-scale data breach | Trivial and straightforward |
| High | 7.0-8.9 | Elevated privilege access, significant data loss or downtime | Easy, vulnerability details or exploit code are publicly available, but may need additional attack vectors (e.g., social engineering) |
| Medium | 4.0-6.9 | Limited access but can still cause loss of tangible assets, which may violate, harm, or impede the org's mission, reputation, or interests. | Difficult, requires a skilled attacker, needs additional attack vectors, attacker must reside on the same network, requires user privileges |
| Low | 0.1-3.9 | Very little impact on an org's business | Extremely difficult, requires local or physical system access |
| Informational | 0.0 | Discloses information that may be of interest to an attacker. | Not exploitable but rather is a weakness that may be useful to an attacker should a higher risk issue be found that allows for a system exploit |

## DISCLAIMER | CERTIK

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

# Elevating Your Web3 Journey

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is the largest blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.