Quantum Purse
Audit Report

Mon Dec 15 2025

contact@bitslab.xyz https://twitter.com/scalebit_

ScaleBit

Quantum Purse Audit Report

1 Executive Summary

1.1 Project Information

Description	Quantum Purse is a Web3 wallet for CKB.
Type	Wallet
Auditors	jolyon, poetyellow, fishmen
Timeline	Wed Nov 19 2025 - Mon Dec 15 2025
Languages	Javascript, Typescript
Platform	CKB
Methods	Dependency Check, Fuzzing, Static Analysis, Manual Review

1.2 Files in Scope

The following are the directories of the original reviewed files.

Directory
https://github.com/tea2x/quantum-purse/src

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	9	9	0
Critical	0	0	0
Major	1	1	0
Medium	1	1	0
Minor	2	2	0
Informational	5	5	0

1.4 ScaleBit Audit Breakdown

ScaleBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Integer overflow/underflow
- Infinite Loop
- Infinite Recursion
- Race Condition
- Traditional Web Vulnerabilities
- Memory Exhaustion Attack
- Disk Space Exhaustion Attack
- Side-channel Attack
- Denial of Service
- Replay Attacks
- Double-spending Attack
- Eclipse Attack
- Sybil Attack
- Eavesdropping Attack
- Business Logic Issues
- Contract Virtual Machine Vulnerabilities
- Coding Style Issues

1.5 Methodology

Our security team adopted **"Dependency Check"**, **"Automated Static Code Analysis"**, **"Fuzz Testing"**, and **"Manual Review"** to conduct a comprehensive security test on the code in a manner closest to real attacks. The main entry points and scope of the security testing are specified in the **"Files in Scope"**, which can be expanded beyond the scope according to actual testing needs. The main types of this security audit include:

(1) Dependency Check

A comprehensive check of the software's dependency libraries was conducted to ensure all external libraries and frameworks are up-to-date and free of known security vulnerabilities.

(2) Automated Static Code Analysis

Static code analysis tools were used to find common programming errors, potential security vulnerabilities, and code patterns that do not conform to best practices.

(3) Fuzz Testing

A large amount of randomly generated data was inputted into the software to try and trigger potential errors and exceptional paths.

(4) Manual Review

The scope of the code is explained in section 1.2.

(5) Audit Process

- Clarify the scope, objectives, and key requirements of the audit.
- Collect related materials such as software documentation, architecture diagrams, and lists of dependency libraries to provide background information for the audit.
- Use automated tools to generate a list of the software's dependency libraries and employ professional tools to scan these libraries for security vulnerabilities, identifying outdated or known vulnerable dependencies.
- Select and configure automated static analysis tools suitable for the project, perform automated scans to identify security vulnerabilities, non-standard coding, and

potential risk points in the code. Evaluate the scanning results to determine which findings require further manual review.

- Design a series of fuzz testing cases aimed at testing the software's ability to handle exceptional data inputs. Analyze the issues found during the testing to determine the defects that need to be fixed.
- Based on the results of the preliminary automated analysis, develop a detailed code review plan, identifying the focus of the review. Experienced auditors perform line-by-line reviews of key components and sensitive functionalities in the code.
- If any issues arise during the audit process, communicate with the code owner in a timely manner. The code owners should actively cooperate (this may include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- Necessary information during the audit process will be well documented in a timely manner for both the audit team and the code owner.

2 Summary

This report has been commissioned by Quantum Purse with the objective of identifying any potential issues and vulnerabilities within the source code of the Quantum Purse repository, as well as in the repository dependencies that are not part of an officially recognized library. In this audit, we have employed the following techniques to identify potential vulnerabilities and security issues:

(1) Dependency Check

A comprehensive analysis of the software's dependency libraries was conducted using the dependency check tool.

(2) Automated Static Code Analysis

The code quality was examined using a code scanner.

(3) Fuzz Testing

Based on the fuzz tool and by writing harnesses.

(4) Manual Code Review

Manually reading and analyzing code to uncover vulnerabilities and enhance overall quality.

During the audit, we identified 9 issues of varying severity, listed below.

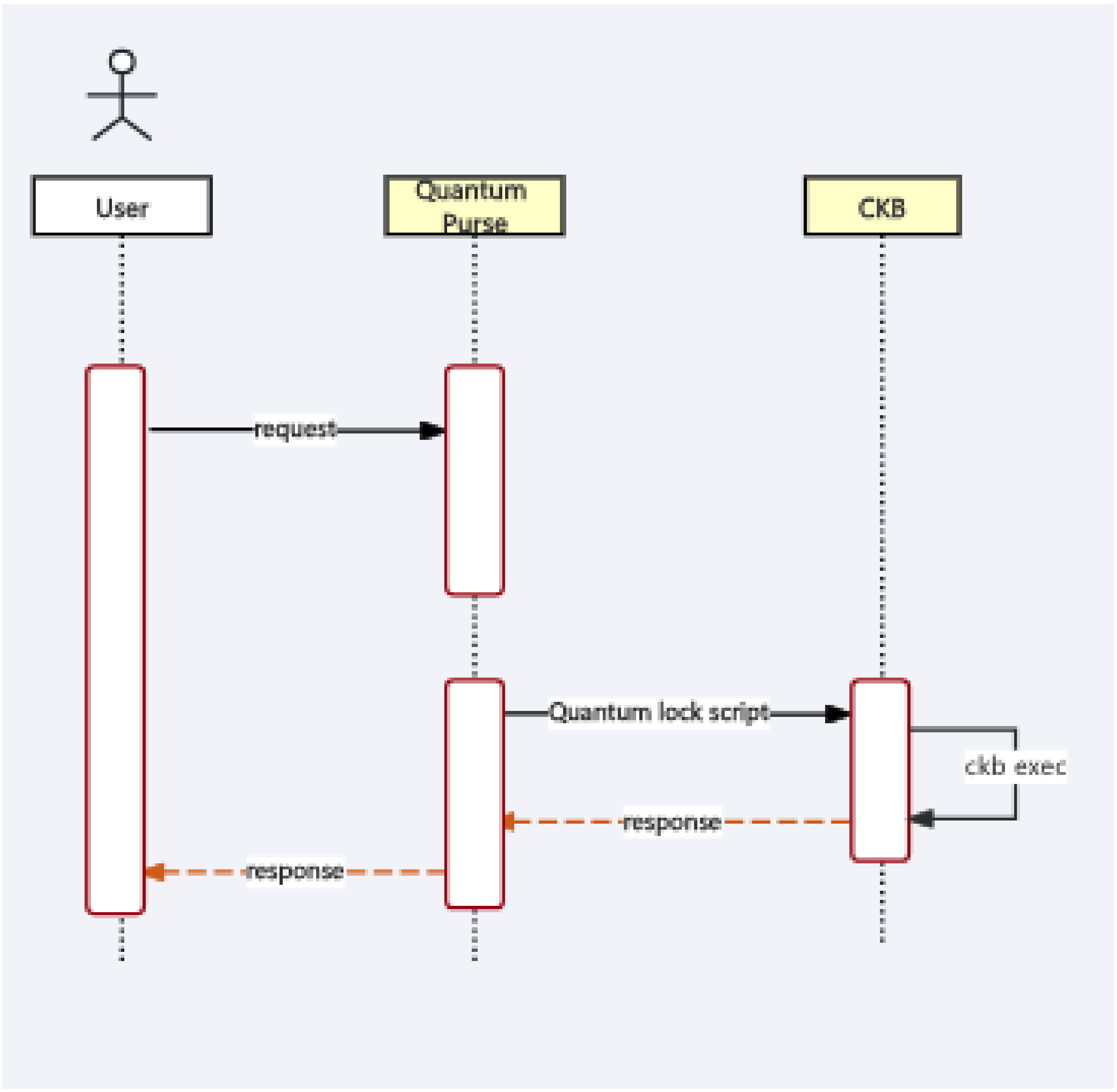
ID	Title	Severity	Status
AUT-7	Password Exposure Due to Residual Memory Retention	Major	Fixed
IND-3	Dependency Security Issues	Informational	Fixed
IND-9	Redux DevTools Potentially Enabled in Production	Informational	Fixed

MAI-6	External Links Opened Without Validation	Informational	Fixed
QPU-1	Limited-Length Random Request ID Introduces Collision Possibility	Minor	Fixed
SEN-4	Inconsistent handling of decimal amounts in send and deposit transactions	Minor	Fixed
WPR-8	Potential Leakage of Sensitive Information via Console Logging	Informational	Fixed
QPU-11	Insecure Random Number Generation	Medium	Fixed
IND1-10	Missing Content Security Policy (CSP) in Main HTML Entry	Informational	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the Quantum Purse repository :

The flowchart for user interactions with ownership wallets using `Quantum lock script` and on the CKB is as follows:



4 Findings

AUT-7 Password Exposure Due to Residual Memory Retention

Severity: Major

Discovery Methods: Manual Review

Status: Fixed

Code Location:

src/ui/components/authentication/authentication.tsx#92

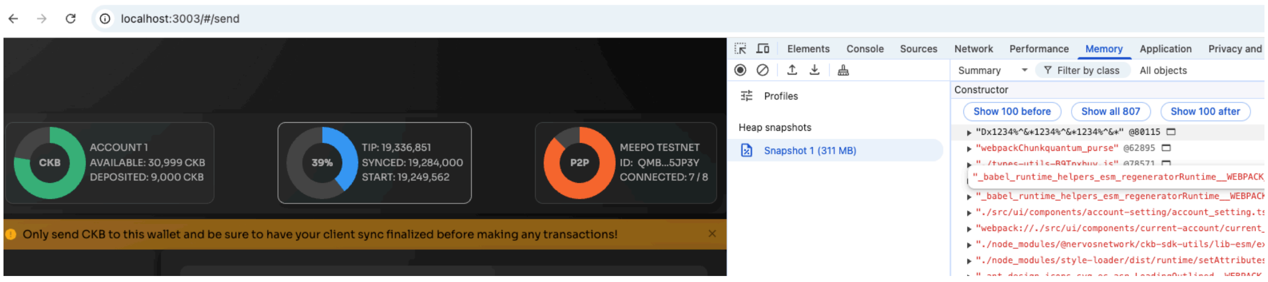
Descriptions:

In the wallet, the user-entered password is not properly cleared from memory after users have used it once.

This behavior exposes the user's password to unintended disclosure, significantly increasing the risk of wallet compromise.

There are 2 types of positions that expose the password

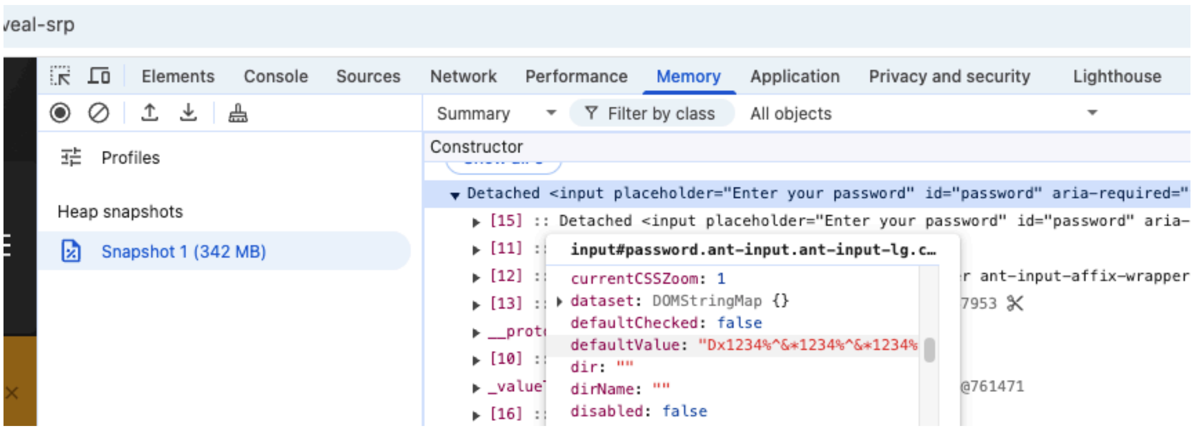
- Password in a javascript string:



Only send CKB to this wallet and be sure to have your client sync finalized before making any transactions!

```
▸ "Dx1234%^&*1234%^&*1234%^&*" @80115
▸ "webpackChunkquantum_purse" @62895
▸ "./types-utils-B9Tpxbuy.js" @78571
 "_babel_runtime_helpers_esm_regeneratorRuntime__WEBPACK
▸ "_babel_runtime_helpers_esm_regeneratorRuntime__WEBPACK
▸ "./src/ui/components/account-setting/account_setting.ts
▸ "webpack://./src/ui/components/current-account/current_
▸ "./node_modules/@nervosnetwork/ckb-sdk-utils/lib-esm/ex
▸ "./node_modules/style-loader/dist/runtime/setAttributes
```

- Password in a "Detached <input>"



```
▾ Detached <input placeholder="Enter your password" id="password" aria-required="
  ▸ [15] :: Detached <input placeholder="Enter your password" id="password" aria-
  ▸ [11] ::  input#password.ant-input.ant-input-lg.c…
  ▸ [12] ::  currentCSSZoom: 1                              r ant-input-affix-wrapper
  ▸ [13] :: ▸ dataset: DOMStringMap {}                      7953
  ▸ __prot   defaultChecked: false
  ▸ [10] ::  defaultValue: "Dx1234%^&*1234%^&*1234%
  ▸ _value   dir: ""                                        @761471
  ▸ [16] ::  dirName: ""
             disabled: false
```

Suggestion:

It's recommended to fill the password string with 0 after using it.

Resolution:

This issue has been fixed in "https://github.com/tea2x/quantum-purse/pull/93". The client use a "uint8array" to store and clean password, and use a "HTML <intput>" tag to avoid React caching mechanism.

IND-3 Dependency Security Issues

Severity: Informational

Discovery Methods: Dependency Check

Status: Fixed

Code Location:

src/ui/store/index.ts#1

Descriptions:

The project currently depends on several packages with known security vulnerabilities:

- **glob 10.2.0 – 10.4.5**: Command injection via `-c/--cmd` in CLI (`high`) – GHSA-5j98-mcp5-4vw2
- **js-yaml 4.0.0 – 4.1.0**: Prototype pollution in `merge` (`moderate`) – GHSA-mh29-5h37-fv8m
- **node-forge <=1.3.1**: Multiple ASN.1 parsing vulnerabilities (`high`) – GHSA-554w-wpv2-vw27, GHSA-65ch-62r8-g69g, GHSA-5gfm-wpxj-wjgq
- **tar 7.5.1**: Race condition potentially exposing uninitialized memory (`moderate`) – GHSA-29xp-372q-xqph

Suggestion:

It is recommended to update these dependencies to fix versions.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

IND-9 Redux DevTools Potentially Enabled in Production

Severity: Informational

Discovery Methods: Manual Review

Status: Fixed

Code Location:

src/ui/store/index.ts#11

Descriptions:

The Redux store is initialized without explicitly configuring DevTools options, which may result in Redux DevTools being available in production builds. This can expose the full application state, including wallet-related data, to anyone with access to Electron DevTools.

`src/ui/store/index.ts` :

```
export const store = init<RootModel, FullModel>({
  models,
  plugins: [
    loadingPlugin({
      asNumber: false,
    }),
  ],
});
```

If Redux DevTools is enabled in production, an attacker (or any user) can:

1. Open Electron DevTools
2. Switch to the Redux tab
3. Inspect the entire state tree, for example:

```
{
  wallet: {
    accounts: [...],   // all account information
    current: {...},    // current account
    syncStatus: {...}, // synchronization status
  }
}
```

They may also perform time-travel debugging and export the state, which increases the risk of sensitive information leakage and can assist in further attacks.

Suggestion:

It's recommended to ensure that in production builds Redux DevTools is disabled and cannot connect to the application state.

Resolution:

This issue has been fixed. The client has adopted our suggestions in the `7992b68c89fd3351664c26253333a7638921ebfa` commit.

MAI-6 External Links Opened Without Validation

Severity: Informational

Discovery Methods: Manual Review

Status: Fixed

Code Location:

main.js#43

Descriptions:

The Electron main process `setWindowOpenHandler` directly calls `shell.openExternal` for all URLs without protocol or domain validation, allowing arbitrary URLs to be opened.

`main.js` :

```
mainWindow.webContents.setWindowOpenHandler(({ url }) => {
  shell.openExternal(url);
});
```

Attack Requirements

There should be a XSS flaw in the app to exploit this issue.

Possibale Attack

Malicious code can open local files using `file://` protocol (e.g., `file:///etc/passwd`)

Suggestion:

Only allow `https:` and `http:` protocols, explicitly block `file:` , `javascript:` , and other dangerous protocols.

Resolution:

This issue has been fixed. The client has adopted our suggestions in the `0f5408382ab05b0ed19d894b526fd0a2ceb4371d` commit.

QPU-1 Limited-Length Random Request ID Introduces Collision Possibility

Severity: Minor

Discovery Methods: Manual Review

Status: Fixed

Code Location:

src/core/quantum_purse.ts#93

Descriptions:

The function `sendRequestToWorker` generates requestId values using `Math.random().toString(36).substring(7)`, which typically produces a 7-character string composed of digits and lowercase letters. While the single-instance collision probability is low, the limited entropy increases the chance of collisions over long-running sessions or high request volumes.

Suggestion:

It is recommended to ues more collision-resistant method, such as using UUID, or alternatively increasing the length of the generated identifier.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

SEN-4 Inconsistent handling of decimal amounts in send and deposit transactions

Severity: Minor

Discovery Methods: Manual Review

Status: Fixed

Code Location:

src/ui/pages/Send/Send.tsx#255

Descriptions:

Currently, in the wallet, the input for **send** and **deposit** transactions only accepts integer values. If a user inputs a decimal value, it triggers an error during conversion to `BigInt`. However, when `isSendMax` is set to `true`, the wallet supports transactions with decimal units. This behavior creates an inconsistency in the handling of decimal amounts across different transaction modes.

Suggestion:

Standardize the input handling for decimal amounts across all transaction types.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

WPR-8 Potential Leakage of Sensitive Information via Console Logging

Severity: Informational

Discovery Methods: Manual Review

Status: Fixed

Code Location:

webpack/webpack.prod.js#1

Descriptions:

The application contains multiple `console.log` and `console.error` statements that remain active in production builds. These logs may expose internal error objects, application state, or other sensitive operational details to users or to log collectors.

Examples include logging internal errors and scanner error messages:

```
// Example: logging light client initialization failures
console.error("Failed to start light client:", error);

// Example: logging QR scanner errors
console.log(errorMessage);
```

In a production Electron environment, such logs can be viewed via DevTools, captured by external log aggregation systems, or included in crash reports. This increases the risk that stack traces, internal configuration, or runtime state are exposed, which can help attackers understand the application’s internal design, identify other weaknesses, or access potentially sensitive information included in error messages.

Suggestion:

It's recommended to configure the production build to strip console calls.

Resolution:

This issue has been fixed. The client has adopted our suggestions in the `151f5a4a903b6fa18c97f19defa3cc2f7d1300fe` commit.

QPU-11 Insecure Random Number Generation

Severity: Medium

Discovery Methods: Manual Review

Status: Fixed

Code Location:

src/core/quantum_purse.ts#93;

public/status.worker.js#7

Descriptions:

```
private sendRequestToWorker(command: string): Promise<any> {
    if (!this.worker) throw new Error("Worker not initialized");
    return new Promise((resolve, reject) => {
      const requestId = Math.random().toString(36).substring(7);
      this.pendingRequests.set(requestId, { resolve, reject });
      this.worker!.postMessage({ command, requestId });
    });
  }
```

The `requestId` in the code is generated using `Math.random()`. This random number generation scheme lacks cryptographic security and carries a probability of generating two identical `requestId` values consecutively. This can cause original messages to be overwritten, leading to a series of functional issues.

Suggestion:

Use a more secure random number generation algorithm.

Resolution:

Developer reply It is used for the communication of a status updater cron job. Not the wallet seed.

IND1-10 Missing Content Security Policy (CSP) in Main HTML Entry

Severity: Informational

Discovery Methods: Manual Review

Status: Fixed

Code Location:

public/index.html#7

Descriptions:

The main HTML entry file does not define any Content Security Policy (CSP). Without CSP, the application lacks a strong client-side defense layer against cross-site scripting (XSS), injection of untrusted scripts, and unauthorized resource loading. `public/index.html` :

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Quantum Purse</title>
  <!-- No CSP defined here -->
</head>
```

Suggestion:

It's recommended to add a restrictive CSP meta tag.

Resolution:

This issue has been fixed. The client has adopted our suggestions in `0f5408382ab05b0ed19d894b526fd0a2ceb4371d` commit.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information or assets at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information or assets at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.