Veridise

Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

KADENA

Kadena Snap

Veridise Inc.

Sep. 25, 2024

► **Prepared For:**

Kadena
https://kadena.io/

► **Prepared By:**

Bryan Tan
Jacob Van Geffen

► **Contact Us:**

contact@veridise.com

► **Version History:**

Sep. 25, 2024	V3 - Added Kadena Snap publication information by request of the developers
Sep. 17, 2024	V2 - Updated issue statuses in response to developer fixes
Sep. 12, 2024	V1

Contents

1 Executive Summary

From Sep. 9, 2024 to Sep. 11, 2024, Kadena engaged Veridise to conduct a security assessment of the source code of what would eventually become version 1.0.2 of Kadena Snap. Specifically, the security assessment covered an update to a closed-source version of the snap that was previously reviewed by Veridise security analysts*. Compared to the previous version, the new version simplifies the account management logic and introduces support for storing hardware accounts. Veridise conducted the assessment over 6 person-days, with 2 security analysts reviewing the project over 3 days on commit `dcf0478d` of the closed-source repository. The review strategy involved a manual code review of the program source code performed by Veridise security analysts.

Following the security assessment, the Kadena Snap developers migrated the code to the open source `kadena-community/kadena.js` repository on GitHub and released the snap as the npm package `@kadena/snap@v1.0.2`. The Veridise security analysts have confirmed that the Git commit indicated by npm's provenance data, `16c65fe15e6727cb9a03b8a0fd46166ebb02fadb`, corresponds to the code that was reviewed, except for some minor cosmetic changes to the confirmation dialog for signing transactions.

Code Assessment. The Kadena Snap developers provided the source code of the Kadena Snap for the code review. The source code appears to be mostly original code written by the Kadena Snap developers. It contains some documentation in the form of READMEs and documentation comments on functions.

The source code contained a test suite, which the Veridise security analysts noted tests the major workflows of the snap in a variety of scenarios, using both valid and invalid inputs.

Summary of Issues Detected. The security assessment uncovered 2 issues, comprising one 1 warning and 1 informational issue. Specifically, the Veridise analysts identified two bugs that could occur when deleting the currently active network (V-KDS2-VUL-001), as well as a code duplication issue in the account management logic (V-KDS2-VUL-002).

The Kadena Snap developers have fixed all of the identified issues as of commit `5dfa1d0` of the closed-source repository.

Disclaimer. We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

* The previous audit report, if it is publicly available, can be found on Veridise's website at `https://veridise.com/audits/`

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
Kadena Snap (audited)	`dcf0478d`	TypeScript	MetaMask Snaps
Kadena Snap 1.0.2	`16c65fe1`	TypeScript	MetaMask Snaps

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Sep. 9–Sep. 11, 2024	Manual & Tools	2	6 person-days

Table 2.3: Vulnerability Summary.

Name	Number	Acknowledged	Fixed
Critical-Severity Issues	0	0	0
High-Severity Issues	0	0	0
Medium-Severity Issues	0	0	0
Low-Severity Issues	0	0	0
Warning-Severity Issues	1	1	1
Informational-Severity Issues	1	1	1
TOTAL	2	2	2

Table 2.4: Category Breakdown.

Name	Number
Logic Error	1
Maintainability	1

3 Security Assessment Goals and Scope

3.1 Security Assessment Goals

The engagement was scoped to provide a security assessment of Kadena Snap's implementation. During the assessment, the security analysts aimed to answer questions such as:

- Are the account data structures updated correctly when an account is added or deleted?
- Is the user's authorization gained before performing sensitive operations such as account deletion?
- Are there any notable differences between managing normal accounts and hardware accounts?

3.2 Security Assessment Methodology & Scope

Security Assessment Methodology. To address the questions above, the security assessment involved a manual code review performed by human experts.

Scope. The scope of this security assessment is limited to the non-test code contained in the `packages/snap` folder of the source code provided by the Kadena Snap developers, which contains the TypeScript implementation of the Kadena Snap.

During the security assessment, the Veridise security analysts referred to the test files (which are excluded from the scope) to understand the intended behavior of the Kadena Snap.

Methodology. Veridise security analysts reviewed the reports of previous audits for Kadena Snap, inspected the provided tests, and read the Kadena Snap documentation. They then began a review of the code.

3.3 Classification of Vulnerabilities

When Veridise security analysts discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise.

The severity of a vulnerability is evaluated according to the Table 3.1.

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

The likelihood of a vulnerability is evaluated according to the Table 3.2.

The impact of a vulnerability is evaluated according to the Table 3.3:

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

4 Vulnerability Report

This section presents the vulnerabilities found during the security assessment. For each issue found, the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.) is specified. Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-KDS2-VUL-001	Bugs when deleting active network	Warning	Fixed
V-KDS2-VUL-002	Account management code duplication	Info	Fixed

4.1 Detailed Description of Issues

4.1.1 V-KDS2-VUL-001: Bugs when deleting active network

Severity	Warning	**Commit**	dcf0478
Type	Logic Error	**Status**	Fixed
File(s)	`src/services/deleteNetwork.ts`		
Location(s)	deleteNetwork()		
Confirmed Fix At	5dfa1d0		

The `kda_deleteNetwork` snap method allows a user to request a network to be removed from the snap. This is implemented by the `deleteNetwork()` function, which will remove the network with the user-specified ID from the snap state.

```
const network = snapApi.state.networks.find((network) => network.id === id);

if (!network) {
  throw new InvalidRequestError('Network does not exist');
}

const confirm = await snap.request({
  method: 'snap_dialog',
  params: /* ... confirmation dialog code ... */
);

if (!confirm) {
  throw new UserRejectedRequestError('Rejected delete network');
}

const networkIndex = snapApi.state.networks.findIndex(
  (network) => network.id === id,
);
const newState = produce(snapApi.state, (draft) => {
  draft.networks.splice(networkIndex, 1);
});

await snapApi.wallet.request({
  method: 'snap_manageState',
  params: {
    operation: 'update',
    newState,
  },
});
```

Snippet 4.1: Snippet from `deleteNetwork()`

There are two bugs in this implementation:

1. The deletion is still allowed if the active network is the network requested for deletion. However, the active network is never updated, so the "active network" will point to an invalid network ID afterwards.

2. The last network (which would also be the active network) can be deleted. This will put the snap into an inconsistent state where the active network will be invalid until the user creates a new network and sets it as the active network.

Impact If the active network is deleted, then the value of `activeNetwork` in the snap state will refer to a network that no longer exists. This may be confusing to the user. Functionality-wise, this currently does not have any impact; the only place where `activeNetwork` is used is in `signTransaction()`, which will validate that the network specified by the user matches an existing network in the snap state. However, future snap methods that use the `activeNetwork` field may be affected by the two bugs described above.

Recommendation

- Consider changing `deleteNetwork()` to prevent the active network from being deleted. If it is intended to allow the active network to be deleted, add logic to update the active network to another network.
- Reject the `kda_deleteNetwork` request if the user is requesting the last network to be deleted.

Developer Response The developers have applied the recommendation.

4.1.2 V-KDS2-VUL-002: Account management code duplication

Severity	Info	**Commit**	dcf0478
Type	Maintainability	**Status**	Fixed
File(s)	See description		
Location(s)	See description		
Confirmed Fix At	5dfa1d0		

In the snap, hardware and non-hardware accounts are stored separately. For performing various management operations related to these accounts (such as adding, renaming, or deleting accounts), the snap uses different functions depending on whether the affected account is a hardware account or a non-hardware account. For example, to delete an account, the functions `deleteAccount` and `deleteHardwareAccount` are used for deleting non-hardware and hardware accounts, respectively.

For a given operation, these functions use nearly identical code. The `deleteAccount` and `deleteHardwareAccount` methods, for example, only differ in a few lines where `accounts` is switched for `hardwareAccounts`. Duplicating code in this way is dangerous, as developers may fail to propagate necessary changes to both copies of the code.

Impact As the account management functionality changes, developers may fail to update both copies of the duplicated code. This may increase the likelihood of introducing bugs into the code.

Recommendation Extract duplicated code into one or two helper functions that are called by methods that update hardware and non-hardware accounts. The specific pairs of functions that should be refactored in this way are:

- `deleteAccount` and `deleteHardwareAccount`
- `setAccountName` and `setHardwareAccountName`
- `storeAccount` and `storeHardwareAccount`

Developer Response The developers have applied the recommendation.