Veridise Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Web3 Antivirus Snap

Veridise Inc.

October 20, 2023

▸ **Prepared For:**

Web3 Antivirus
https://web3antivirus.io/

▸ **Prepared By:**

Jacob Van Geffen
Bryan Tan

▸ **Contact Us:** contact@veridise.com

▸ **Version History:**

Oct. 20, 2023 V1

Contents

1 Executive Summary

From Oct. 16, 2023 to Oct. 18, 2023, Web3 Antivirus engaged Veridise to review the security of their Web3 Antivirus Snap. The review covered a MetaMask snap that displays security scanning results from an API provided by Web3 Antivirus when the user is about to sign a transaction. Veridise conducted the assessment over 6 person-days, with 2 engineers reviewing code over 3 days on commit `1fdefcd`. The auditing strategy involved manual code review of the source code performed by Veridise engineers.

Code assessment. The Web3 Antivirus Snap developers provided the source code of the Web3 Antivirus Snap contracts for review*. The code appears to be mostly original code written by the Web3 Antivirus Snap developers. No documentation was provided to the Veridise auditors for the audit. Although the auditors attempted to understand the intended behavior of the code from the source code and the publicly available documentation on Web3 Antivirus's website, they were unable to accurately assess whether the snap correctly uses the API it interacts with. The source code contains a small test suite with partial coverage of the code's behaviors.

Summary of issues detected. The audit uncovered 4 issues, consisting of 2 low-severity issues and 2 warnings. Most of the issues relate to the error handling in the snap. Specifically, an HTTP response error may result in the misleading "All good!" message (V-W3AS-VUL-002) being displayed to the user (V-W3AS-VUL-001), the snap error message is very vague (V-W3AS-VUL-004), and there is a backwards compatibility concern with how the snap UI items are rendered (V-W3AS-VUL-003). The Web3 Antivirus Snap developers have resolved all of the reported issues.

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.

* Publicly accessible at `https://github.com/web3-antivirus/web3-antivirus-snap`

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
Web3 Antivirus Snap	`1fdefcd`	TypeScript	MetaMask Snap

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Oct. 16 - Oct. 18, 2023	Manual & Tools	2	6 person-days

Table 2.3: Vulnerability Summary.

Name	Number	Resolved
Critical-Severity Issues	0	0
High-Severity Issues	0	0
Medium-Severity Issues	0	0
Low-Severity Issues	2	2
Warning-Severity Issues	2	2
Informational-Severity Issues	0	0
TOTAL	4	4

Table 2.4: Category Breakdown.

Name	Number
Logic Error	2
Usability Issue	2

3 Audit Goals and Scope

3.1 Audit Goals

The engagement was scoped to provide a security assessment of Web3 Antivirus Snap's implementation. In our audit, we sought to answer questions such as:

- Is the API request constructed in a way that can result in query injection vulnerabilities?
- Are errors correctly handled by the snap?
- Does the snap present the results of the security scan in a way that is appropriate for the user?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions above, our audit involved manual code review by a team of human experts.

Scope. The scope of this audit is limited to the `packages/snap` folder of the source code provided by the Web3 Antivirus Snap developers, which contains the snap part of the Web3 Antivirus Snap.

Methodology. Veridise auditors inspected the provided tests and read the resources available on Web3 Antivirus's website. They then began a manual code review of the code in the scope of the audit.

3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

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

In this case, we judge the likelihood of a vulnerability as follows in Table 3.2:

In addition, we judge the impact of a vulnerability as follows in 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

Vulnerability Report 4

In this section, we describe the vulnerabilities found during our audit. For each issue found, we log the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.). Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-W3AS-VUL-001	Misleading message displayed on API request erro	Low	Fixed
V-W3AS-VUL-002	No-results message suggests that there are no f...	Low	Fixed
V-W3AS-VUL-003	Unsupported UI elements are silently dropped	Warning	Fixed
V-W3AS-VUL-004	Error message needs additional details	Warning	Fixed

4.1 Detailed Description of Issues

4.1.1 V-W3AS-VUL-001: Misleading message displayed on API request error

Severity	Low	Commit	1fdefcd
Type	Logic Error	Status	Fixed
File(s)	`w3a/api.ts`, `index.ts`		
Location(s)	getTransactionAnalyze()		
Confirmed Fix At	1f223b3		

The `getTransactionAnalyze()` function is used to query the Web3 Antivirus API endpoint for details on the transaction. If the HTTP response has a status code other than a 2XX status code, then the function returns `null`. However, in the code in `onTransaction()` that calls `getTransactionAnalyze()`, a `null` response will cause the panel built by `getStandardPanel()` to be displayed, rather than an error message.

```
export const getTransactionAnalyze = async (
  transaction: { [key: string]: Json },
  chainId: string,
  transactionOrigin?: string,
): Promise<SnapResponseDTO | null> => {
  // ...
  const response = await fetch(/* ... */);
  if (response.ok) {
    const result = (await response.json()) as SnapResponseDTO;
    return result;
  }
  return null;
};
```

Snippet 4.1: Relevant lines in `getTransactionAnalyze()`

```
const analyze = await getTransactionAnalyze(
  transaction,
  chainId,
  transactionOrigin
);

if (!analyze?.items?.length) {
  return {
    content: getStandardPanel(),
  }
}
```

Snippet 4.2: Relevant lines in `onTransaction()` that handle the error incorrectly.

Impact If the HTTP request fails, then the snap will show a message saying that there aren't "any risks in this transaction", even if the transaction may actually have risks. This is especially problematic due to V-W3AS-VUL-002.

```
export const getStandardPanel = (): Panel =>
  panel([
    heading("All good! "),
    text(
      "W3A hasn't detected any risks in this transaction. You can proceed worry-
    free."
    ),
  ]);
```

Snippet 4.3: Definition of `getStandardPanel()`

Recommendation Either 1) throw an error if the HTTP request fails, or 2) change the error handling code so that it checks whether `analyze` is null before accessing the `.items?.length` (e.g., this can be easily done by changing `analyze?.items` to `analyze!.items` to trigger an error, which will be caught by the try-catch block).

Developer Response The developer changed `getTransactionAnalyze()` to throw an error.

4.1.2 V-W3AS-VUL-002: No-results message suggests that there are no false negatives

Severity	Low	Commit	1fdefcd
Type	Usability Issue	Status	Fixed
File(s)	panels.ts		
Location(s)	getStandardPanel()		
Confirmed Fix At	1f223b3		

The `getStandardPanel()` function is used to display a message when the Web3 Antivirus API endpoint does not find any risks associated with the transaction. However, the phrasing of the message may cause users to let their guard down. Specifically, the "You can proceed worry-free." assumes that there are no false negatives associated with the API endpoint. This may not be true if the transaction is part of a social engineering attack or a multi-step attack, for example.

```
export const getStandardPanel = (): Panel =>
    panel([
      heading("All good! "),
      text(
        "W3A hasn't detected any risks in this transaction. You can proceed worry-
    free."
      ),
    ]);
```

Snippet 4.4: Definition of `getStandardPanel()`

Impact The message may cause users to incorrectly believe that they are safe when they are in an unsafe situation.

Recommendation Remove the "You can proceed worry-free" from the message, and consider inserting a message that reminds the user to always watch out for signs of scams or fraud.

Developer Response The developers changed the message to "W3A found no risks, but don't drop your guard and watch out for any suspicious activity."

4.1.3 V-W3AS-VUL-003: Unsupported UI elements are silently dropped

Severity	Warning	Commit	1fdefcd
Type	Logic Error	Status	Fixed
File(s)	w3a/utils.ts		
Location(s)	renderLayoutFromSnapResponse()		
Confirmed Fix At	1f223b3		

The `renderLayoutFromSnapResponse()` renders a JSON object returned by the API endpoint as a snap UI item. It does this by mapping a list of supported component identifiers (contained in `LAYOUT_COMPONENT_BY_TYPE`) to the actual `@metamask/snaps-ui` functions used to construct the components. However, if a component is not contained in `LAYOUT_COMPONENT_BY_TYPE`, then the `component` variable will be undefined. Thus, only the currently accumulated items `acc` will be returned (i.e., the `component` will be excluded from the list of UI items returned by `renderLayoutFromSnapResponse`). No warning or error will be shown to the user if this occurs.

```
const LAYOUT_COMPONENT_BY_TYPE = {
  [NODE_TYPE.COPYABLE]: copyable,
  [NODE_TYPE.DIVIDER]: divider,
  [NODE_TYPE.HEADING]: heading,
  [NODE_TYPE.SPINNER]: spinner,
  [NODE_TYPE.TEXT]: text,
};

export const renderLayoutFromSnapResponse = (
  items: SnapResponseDTOItem[]
): Component[] =>
  items.reduce<Component[]>((acc, { node, data }) => {
    const component = LAYOUT_COMPONENT_BY_TYPE[node];
    if (component) {
      return [...acc, data ? component(data) : component()];
    }
    return acc;
  }, []);
```

Snippet 4.5: Relevant functions

Note that the return value of `renderLayoutFromSnapResponse()` is directly used to construct the panel in `onTransaction()`, implying that the backend is responsible for rendering the snap panel.

```
const analyze = await getTransactionAnalyze(/* ... */);
// ...
const panelData = renderLayoutFromSnapResponse(analyze.items);

return {
  content: panel(panelData),
};
```

Snippet 4.6: Relevant code in `onTransaction()`

Impact This issue mainly affects backwards compatibility. If the API endpoint used by the snap is updated to return new UI item types, then older versions of the snap will not show those items. This may result in the snap failing to work as intended.

Recommendation To alleviate backwards compatibility concerns, move the code that renders the snap panel to the snap itself rather than rendering the snap panel in the API backend.

Developer Response The developers do not intend to implement the recommendation for the following reason:

> It was a primary goal for us to transfer rendering components to the backend. This allows us to avoid redeploying the plugin for new features.

The auditors instead recommend that when a component type cannot be found, then either (1) the component should be replaced with an error panel, or (2) an error should be thrown. The developers applied the first alternative recommendation.

4.1.4 V-W3AS-VUL-004: Error message needs additional details

Severity	Warning	Commit	1fdefcd
Type	Usability Issue	Status	Fixed
File(s)	`index.ts`		
Location(s)	onTransaction()		
Confirmed Fix At	1f223b3		

In the transaction handler, a single common error message is displayed whenever any error occurs.

```
try {
   ...
} catch (error) {
  return {
    content: getErrorPanel(),
  };
}
```

Snippet 4.7: Relevant lines in `onTransaction()` that handle the error incorrectly.

However, there are multiple sources that the error could have originated from. By displaying only a single error message for any error, all information about what could have caused the error is lost to the user.

```
"Something went wrong, but no worries, we are already sorting it out. Thank you
   for your patience!"
```

Snippet 4.8: The message shown in `getErrorPanel()`

Impact Because the snap is meant to alert users of security risks involving transactions, users may assume that the error is related to their account or the transaction rather than with the snap. Furthermore, even if users understand that the error is related to the snap, they will have a difficult time understanding and avoiding the error.

Recommendation

- Modify the error message to clearly indicate that the error is related to the snap itself.
- Add additional information about the type of error caught. Some ways this could be done include:
  1. Perform a try-catch around the individual operations within `onTransaction` (e.g. around the calls to `getTransactionAnalyze` and `renderLayoutFromSnapResponse` separately). This would allow the snap to give different error messages depending on which sub-procedure failed.
  2. Include the type of the error and the internal error message in the popup error message. This solution is less recommended, because though this would help some users understand what went wrong, many users may be confused by this low-level information.

Developer Response The developers added additional error messages in `onTransaction` which will be displayed for errors in `getTransactionAnalyze()` and `renderLayoutFromSnapResponse()`.