Veridise Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

quickintel-snap

Veridise Inc.
October 10, 2023

▸ **Prepared For:**

Quick Intel
https://quickintel.io/

▸ **Prepared By:**

Jacob Van Geffen
Bryan Tan

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

▸ **Version History:**

Oct. 10, 2023 V1

Contents

1 Executive Summary

From Oct. 4, 2023 to Oct. 6, 2023, Quick Intel engaged Veridise to review the security of their quickintel-snap. The review covered a MetaMask snap that displays security information returned by Quick Intel's auditing service 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 `3af7d0b`. The auditing strategy involved manual code review of the source code performed by Veridise engineers.

Code assessment. The quickintel-snap developers provided the source code of the quickintel-snap contracts for review*. The code appears to be a based on the MetaMask snap template†. No documentation was provided to the Veridise auditors for the audit; thus, the auditors attempted to understand the intended behavior of the code from the source code and the publicly available documentation on Quick Intel's website. The source code does not contain a test suite, nor does it have evidence that the developers use linting or static analysis tools.

Summary of issues detected. The audit uncovered 6 issues, 1 of which are assessed to be of high or critical severity by the Veridise auditors. Specifically, a hardcoded API key initialized to the empty string causes the snap to fail (V-QIS-VUL-001). The Veridise auditors also identified a medium-severity issue where the transaction recipient address is not provided to the API request (V-QIS-VUL-002), as well as 3 low-severity issues and 1 warning. The quickintel-snap developers 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/Quick-Intel/quickintel-snap`
† `https://github.com/MetaMask/template-snap-monorepo/`

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
quickintel-snap	`3af7d0b`	TypeScript	MetaMask Snap

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Oct. 4 - Oct. 6, 2023	Manual	2	6 person-days

Table 2.3: Vulnerability Summary.

Name	Number	Resolved
Critical-Severity Issues	1	1
High-Severity Issues	0	0
Medium-Severity Issues	1	1
Low-Severity Issues	3	3
Warning-Severity Issues	1	1
Informational-Severity Issues	0	0
TOTAL	6	6

Table 2.4: Category Breakdown.

Name	Number
Logic Error	3
Data Validation	1
Query Injection	1
Authorization	1

3 Audit Goals and Scope

3.1 Audit Goals

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

- Does the snap provide the correct parameters for invoking the API endpoint?
- Does the snap build links in a way that is vulnerable to query injection attacks?
- Are errors correctly handled by the snap?
- Does the snap have any unnecessary permissions listed in the manifest file?

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 quickintel-snap developers, which contains the snap part of the quickintel-snap.

Methodology. Before the audit, the Veridise auditors met with the quickintel-snap developers to ask questions about the intended behavior of the code. Then the Veridise auditors 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:

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

In addition, we judge the impact of a vulnerability as follows in Table 3.3:

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

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-QIS-VUL-001	Hardcoded API key	Critical	Fixed
V-QIS-VUL-002	API endpoint call is missing recipient address	Medium	Intended Behavior
V-QIS-VUL-003	Incorrect HTTP response error handling for API ...	Low	Fixed
V-QIS-VUL-004	Potential URL query injection when building lin...	Low	Fixed
V-QIS-VUL-005	Chain ID is read from node instead of transacti...	Low	Fixed
V-QIS-VUL-006	Unnecessary RPC permission	Warning	Fixed

4.1 Detailed Description of Issues

4.1.1 V-QIS-VUL-001: Hardcoded API key

Severity	Critical	Commit	3af7d0b
Type	Logic Error	Status	Fixed
File(s)	index.ts		
Location(s)	getData()		
Confirmed Fix At	0ea4556		

An API key is required for the Quick Intel API endpoint that is used to retrieve information about the transaction recipient. However, when the API endpoint is invoked in `getData()`, the snap will use a hardcoded API key in the `QUICK_INTEL_KEY` variable.

```
async function getData(bodydata: object | null): Promise<any> {
  const response = await fetch(`${QUICK_INTEL_URL}`, {
    method: 'POST',
    headers: {
      Accept: 'application/json',
      'Content-Type': 'application/json',
      api_key: QUICK_INTEL_KEY,
    },
    body: JSON.stringify(bodydata),
  });
  return bodydata ? response?.json() : null;
}
```

Snippet 4.1: Definition of `getData()`

Impact If the hardcoded API key is invalid, then the snap will not work correctly. If it is valid, then all users will be able to use the API key to invoke the API endpoint. By default, `QUICK_INTEL_KEY` is set to empty string, in which case the API key will be invalid.

```
const QUICK_INTEL_KEY = '';
```

Snippet 4.2: Definition of `QUICK_INTEL_KEY` in `quickintel/constants.ts`

Recommendation Create an exported method `onRpcRequest()` that allows the user to save a provided API key to persistent storage, so that it can be loaded and used in `getData()`.

Developer Response The developers fixed this issue as follows:

> Read and accepted. Per recommendations, another strategy has been implemented for remidation, as outlined below.
>
> The QUICK_INTEL_KEY, will still be use, but it has been converted to a public API key. To avoid abuse, a rate limit of 5 calls per 10 seconds has been added. Additionally this API key will only work for Snap Audits.

4.1.2 V-QIS-VUL-002: API endpoint call is missing recipient address

Severity	Medium	Commit	3af7d0b
Type	Logic Error	Status	Intended Behavior
File(s)	index.ts		
Location(s)	onTransaction()		
Confirmed Fix At	fbe944b		

To collect transaction insights, the snap makes an API call to collect audit information from Quick Intel for the token involved in the transaction. Although the Veridise auditors were not provided any information about the API endpoint used in the snap, the Quick Intel documentation for token scans seems to suggest that a scan requires as input (1) a chain ID; and (2) a token address. However, the body of the API endpoint request only contains the chain ID and the transaction call data.

```
1 const txData = transaction?.data;
2
3 const body = {
4     chainid: chainId,
5     txdata: txData?.toString(),
6 };
```

Snippet 4.3: Relevant code in `onTransaction()`

Since the request contains the call data `transaction.data` (i.e., ABI-encoded call arguments to the recipient of the transaction) but not the `to` address (i.e., recipient) of the transaction, there is no way for the backend serving the Quick Intel API to determine which token is associated with the transaction.

Impact Since no token address is provided to the Quick Intel API endpoint, it is likely that the API backend is not returning valid token scanning information for the recipient of the transaction. This has consequences including: users being mislead into thinking a malicious token is a different, safe token; safe tokens being shown as malicious tokens, etc.

Recommendation

- Document the behavior of the API endpoint.
- Include the `transaction.to` (i.e., the address of the recipient) in the `body` for the audit request API call.
- Modify the API endpoint to take the address of the transaction recipient into account.

Developer Response The developer noted that this snap is designed to target method calls to DEX contracts, which is why the recipient address is not included in the API call:

> Reviewed. But the recommended method will not work due to the purpose of the Snap to be of best value when users swap on DEX's, the `transaction.to` attribute will not work as this address is typically the address of the DEX router, and contains no token information.

For further enhancement the transaction.from attribute was also added.

The token data that is being scanned against is within the `transaction.data` values. What Quick Intel does is send the the full transacation data via the API request. The Quick Intel API then deconstructs the transaction data, and checks the value for valid token contracts on each respective chain. For each token contract found, which could also be multiple for mulit-route token contracts, the audit is then returned for each.

To avoid innaccuracies, the TokenName, Symbol, and contract address are returned for user confirmation.

We have also created additional documentation to better understand this process below. `https://docs.quickintel.io/metamask-snap/introduction#how-is-this-done`

4.1.3 V-QIS-VUL-003: Incorrect HTTP response error handling for API endpoint

Severity	Low	Commit	3af7d0b
Type	Data Validation	Status	Fixed
File(s)	index.ts		
Location(s)	getData()		
Confirmed Fix At	0ea4556		

The `onTransaction()` function will call the `getData()` function to send an HTTP POST request to the API endpoint that returns audit information about a specified token. However, the HTTP response code is never checked, and an empty response will cause an error to be thrown rather than for an error message to be displayed in the panel.

```
async function getData(bodydata: object | null): Promise<any> {
  const response = await fetch(`${QUICK_INTEL_URL}`, {
    method: 'POST',
    headers: {
      Accept: 'application/json',
      'Content-Type': 'application/json',
      api_key: QUICK_INTEL_KEY,
    },
    body: JSON.stringify(bodydata),
  });
  return bodydata ? response?.json() : null;
}
```

Snippet 4.4: Implementation of `getData()`

Impact If the API endpoint encounters an error but returns a JSON object as its response, then an error will likely be thrown in `onTransaction()`, causing the snap to be terminated. This will cause the snap to not work as intended.

Recommendation

- Change `getData()` to return null or throw an error if `response.ok()` is false (or the status code does not match the expected status code).
- In `onTransaction()`, remove the `else` branch that throws the `Error`, so that the subsequent `panelArr.push(...)` code will be executed instead.

```
const auditReq = await getData(body);
/* ... */
if (auditReq?.length > 0) {
  /* ... */
} else {
  throw new Error(
    `No Audit Results or chain not supported for Token Audit. Contact Quick Intel for more information.`,
  );
}

if (panelArr.length === 0) {
  panelArr.push(
    text(
      `No Audit Results or chain not supported for Token Audit. Contact Quick Intel for more information.`,
    ),
    /* ... */
  );
}
```

Snippet 4.5: Relevant lines in `onTransaction()`

4.1.4 V-QIS-VUL-004: Potential URL query injection when building link for full audit

Severity	Low	Commit	3af7d0b
Type	Query Injection	Status	Fixed
File(s)	index.ts		
Location(s)	onTransaction()		
Confirmed Fix At	0ea4556		

The transaction insight panel for the Quick Intel snap includes a link to the ”full audit” on the Quick Intel website. This link is constructed with string interpolation, but without any sanitization of the query parameters.

```
1 text(`View Full Audit on Quick Intel:`),
2 copyable(
3   `https://app.quickintel.io/scanner?type=token&chain=${item?.chain}&
      contractAddress=${item?.tokenAddress}`,
4 ),
```

Snippet 4.6: Relevant lines in `onTransaction()`

Impact This should have no security impact, assuming that the `item.chain` is numeric and that the `item.tokenAddress` is a valid string of hex digits. However, if the `item.chain` can be an arbitrary string, then this may be vulnerable to a URL parameter query injection attack.

Recommendation Use the `URL` class and its `searchParams` property to build the URL. The `URLSearchParams.append()` method will automatically sanitize the query parameters.

4.1.5 V-QIS-VUL-005: Chain ID is read from node instead of transaction information

Severity	Low	Commit	3af7d0b
Type	Logic Error	Status	Fixed
File(s)	index.ts		
Location(s)	onTransaction()		
Confirmed Fix At	0ea4556		

To determine which `chainId` the transaction is from, the `chainId` will be retrieved by making a network request to the Ethereum node that MetaMask is currently connected to. However, this is unnecessary because one of the `onTransaction` function arguments is the `chainId`.

```
1 export const onTransaction: OnTransactionHandler = async ({ transaction }) => {
2   const quickIntelShield = '';
3
4   const chainId = await ethereum.request({ method: 'eth_chainId' });
5
6   const txData = transaction?.data;
7
8   const body = {
9     chainid: chainId,
```

Snippet 4.7: Relevant lines in `onTransaction()`

Impact The chain ID that is retrieved will correspond to the chain ID of the Ethereum node that MetaMask is currently connected to, rather than the actual chain ID of the transaction. The two may not necessarily be equal, depending on how the user interacts with the MetaMask UI.

Furthermore, the line that retrieves `chainId` is the only place where the Ethereum provider is used; if it is removed, then the `endowment:ethereum-provider` permission can be removed from the snap manifest.

Recommendation

- Add the `chainId` argument to `onTransaction` and use it in the API request body instead. Note that the `chainId` provided in `onTransaction` is in CAIP-2 format (see documentation), rather than a numeric chain ID returned by an Ethereum client.
- Remove the `endowment:ethereum-provider` permission from the snap manifest.

4.1.6 V-QIS-VUL-006: Unnecessary RPC permission

Severity	Warning	**Commit**	3af7d0b
Type	Authorization	**Status**	Fixed
File(s)	`snap.manifest.json`		
Location(s)	N/A		
Confirmed Fix At	b6fc3d7		

The snap manifest lists an `endowment:rpc` permission for `dapps`, meaning that on websites for which the snap is enabled, JavaScript code on the website can send JSON-RPC requests to the snap. However, the snap does not export an `onRpcRequest()` function that handles such requests, so the permission is redundant.

Impact While this should have no security impact, users installing the snap will be required to grant an unnecessary permission to the snap.

Recommendation Remove the entire `endowment:rpc` permission from the manifest.