Veridise. Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

KLEROS

Kleros Scout

Veridise Inc.

June 28, 2023

► **Prepared For:**

Kleros
https://kleros.io/

► **Prepared By:**

Himanshu
Bryan Tan

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

► **Version History:**

Jun. 28, 2023 V1

Contents

1 Executive Summary

From Jun. 12, 2023 to Jun. 14, 2023, Kleros engaged Veridise to review the security of their Kleros Scout, a MetaMask snap that will display metadata from Kleros Curate registries before a user confirms a transaction with MetaMask. This information includes details such as: whether the address has an associated project, whether the current domain is "verified" for that address, and whether the address is a token (and if it is, display its name and symbol). Veridise conducted the assessment over 6 person-days, with 2 engineers reviewing code over 3 days on commit 34d1332. The auditing strategy involved a tool-assisted analysis of the source code performed by Veridise engineers as well as extensive manual auditing.

Code assessment. The Kleros Scout developers provided the source code of the Kleros Scout implementation for review. To facilitate the Veridise auditors' understanding of the code, the Kleros Scout developers provided a list of instructions on how to build and run the snap locally. The source code also contained some documentation in the form of READMEs and documentation comments on functions and storage variables.

The source code does not contain a test suite, but several files in the source code indicate that the developers use linting and static analysis tools such as ESLint.

Summary of issues detected. The audit uncovered 6 issues, with the highest severity issues being 1 issue assessed to be of low severity by the Veridise auditors. Specifically, the regex used to capture the domain part of the originating URL will also capture the port number (V-KLS-VUL-001). The Veridise auditors also identified 2 warnings and 3 informational findings, including an overly-broad GraphQL constraint (V-KLS-VUL-002) and advice to avoid URL query injection attacks for a future link feature (V-KLS-VUL-004).

The Kleros Scout developers resolved all of the issues.

Recommendations. After auditing the protocol, the auditors had a few suggestions to improve the Kleros Scout code. Mainly, the auditors felt uncomfortable that there was no automated testing, and they noted that some parts of the code could be tested independently of the Snap. For example, the `getDomainFromUrl()` and `getInsights()` functions do not depend on any Snap-specific information and could be tested with unit tests. Issues such as V-KLS-VUL-001 could have been caught by unit tests. However, independent unit testing of specific functionality does not preclude integration testing of the whole Snap, as the actual execution environment is locked down.

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.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
Kleros Scout	34d1332	TypeScript	MetaMask Snaps

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Jun. 12 - Jun. 14, 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	1	1
Warning-Severity Issues	2	2
Informational-Severity Issues	3	3
TOTAL	6	6

Table 2.4: Category Breakdown.

Name	Number
Data Validation	2
Logic Error	2
Query Injection	1
Maintainability	1

3 Audit Goals and Scope

3.1 Audit Goals

The engagement was scoped to provide a security assessment of Kleros Scout's source code. In our audit, we sought to answer the following questions:

- Are all untrusted inputs sanitized correctly?
- Does the Snap transmit any information that should be kept private?
- Do the Snap permissions follow the principle of least privilege?
- Is it possible for the insights to incorrectly indicate that the transaction origin is a "verified" domain if it is not contained in the registry?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions above, our audit involved extensive manual code review.

Scope. The scope of this audit is limited to the `packages/snap` folder of the source code provided by the Kleros Scout developers, which contains the smart contract implementation of the Kleros Scout.

Methodology. To understand the intended behavior, the Veridise auditors first read the public documentation available on Klero's website* and followed the developer-provided instructions to run the Snap. They then began a manual audit of the code. During the audit, the Veridise auditors also experimented with the GraphQL endpoint referenced by the source code.

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:

* `https://kleros.gitbook.io/docs`

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

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., acknowleged, fixed, etc.). Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-KLS-VUL-001	URL domain regex captures too much	Low	Fixed
V-KLS-VUL-002	Query constraint for targetAddresses checks con. .	Warning	Fixed
V-KLS-VUL-003	Assumptions about chain ID	Warning	Acknowledged
V-KLS-VUL-004	Query injection pitfalls for future link feature	Info	Fixed
V-KLS-VUL-005	Some values fetched by query are unused	Info	Fixed
V-KLS-VUL-006	Lack of error handling when sending GraphQL qu	Info	Fixed

4.1 Detailed Description of Issues

4.1.1 V-KLS-VUL-001: URL domain regex captures too much

Severity	Low	**Commit**	34d1332
Type	Data Validation	**Status**	Fixed
File(s)	`index.ts`		
Location(s)	getDomainFromUrl()		
Fixed At	40730da		

The `getDomainFromUrl()` function is used to extract the domain from a given URL. However, it is implemented using a regex which captures more information than just the domain. For example, for the input `https://example.com:80/`, the capture group will correspond to `example.com:80` which includes the port number as well.

```
const getDomainFromUrl = (url: string): string | null => {
  const match = url.match(/^https?:\/\/([^/?#]+)(?:[/?#]|$)/iu);
  if (match) {
    return match[1];
  }
  return null;
};
```

Snippet 4.1: Implementation of `getDomainFromUrl()`

Impact Assuming that the transaction origin passed by MetaMask is an arbitrary URL, the extracted domain may not correspond to the actual domain entry that should be checked for in the registry (e.g., if the extracted domain has a port but the registry version does not, or vice versa). Thus, the snap may falsely mark the domain as "verified" or "not verified" when the opposite is true.

Recommendation The developers should clarify whether the "domain" is only intended to only be the domain (e.g., do not include port), or if it is the hostname (e.g., including port). In any case, we recommend that the developers replace the regex with the standard `URL` class instead, which should be compliant with the official WHATWG URL standard. The `URL` class provides fields such as `.hostname` that can be used to reliably extract the parts of the URL that are needed.

Developer Response The developers noted that the registry should not include the port number as part of the domain, so it should not be captured by the regex.

4.1.2 V-KLS-VUL-002: Query constraint for targetAddresses checks containment

Severity	Warning	**Commit**	34d1332
Type	Logic Error	**Status**	Fixed
File(s)	`index.ts`		
Location(s)	fetchGraphQLData()		
Fixed At	baebfab		

To gather the transaction insights, the Snap will make a GraphQL query to a publicly accessible API endpoint. Part of the constraints in the query is to find items whose address key **contains** the target address (case insensitive). This seems like an unnecessarily lax condition.

```
  query($targetAddress: String!, $domain: String!) {
    addressTags: litems(where:{
      registry:"0x66260c69d03837016d88c9877e61e08ef74c59f2",
      key0_contains_nocase: $targetAddress,
      status_in:[Registered, ClearingRequested]
    }, first: 1) {
...
        contractDomains: litems(where:{
      registry:"0x957a53a994860be4750810131d9c876b2f52d6e1",
      key0_contains_nocase: $targetAddress,
      key1: $domain,
...
        tokens: litems(where:{
      registry:"0x70533554fe5c17caf77fe530f77eab933b92af60",
      key0_contains_nocase: $targetAddress,
```

Snippet 4.2: Relevant parts of the GraphQL query

Impact Currently, this does not seem to have an impact because addresses on EIP155-supporting blockchains appear to have the same length.

If the developers intend to extend the snap to support a blockchain which has addresses with dynamic lengths, the query could incorrectly retrieve addresses that do not exactly match the target address.

Recommendation Change the query to use an exact match (case insensitive) instead of only checking containment.

Developer Response The developers stated that the only case-insensitive queries that are supported are starts with, ends with, and contains. They have changed the query to check `key0_starts_with_nocase` and `key0_ends_with_nocase` instead as a stopgap.

4.1.3 V-KLS-VUL-003: Assumptions about chain ID

Severity	Warning	**Commit**	34d1332
Type	Data Validation	**Status**	Acknowledged
File(s)	`index.ts`		
Location(s)	onTransaction()		
Fixed At	N/A		

The `onTransaction()` function is invoked when the user is about to perform a transaction. One of the arguments to `onTransaction()` is a CAIP-2 chain ID indicating the chain that the transaction will be performed on. The current implementation makes the following assumptions about the chain ID:

- The chain's namespace is `eip155`. This may not be true of all chains that a user may interact with.
- The reference is numeric. While this assumption seems to hold for `eip155`, CAIP-2 generally allows non-numeric chain IDs.

```
1 const numericChainId = parseInt(chainId.split(':')[1], 16);
2 const caipAddress = `eip155:${numericChainId}:${transaction.to as string}`;
```

Snippet 4.3: Relevant lines in `onTransaction()` that demonstrate the assumptions.

Impact

- When the user attempts to perform a transaction on a non-`eip155` chain and the chain ID is numeric, then the target address will be looked up for the `eip155` chain, not the actual chain being run. If there is a collision between addresses, the information for the address on the `eip155` chain, not the actual chain, may be displayed.
- If the chain ID is not numeric, this will result in the `numericChainId` being set to `NaN`, and a query will still be made.

Recommendation If the chain's namespace is not `eip155`, the snap should display an error message saying that the chain is not supported by the snap, and it should not attempt to fetch insights for the transaction.

Developer Response The developers responded:

> This is a snap for MetaMask, and MetaMask can only interact with eip155 chains. Curate uses CAIP-10 because the *rich address* format is wholly chain agnostic, but for what the snap is concerned, `transaction.to` will always be an Ethereum address, the namespace will always be `"eip155"`, and the reference will be numeric. (In the case of the snap, it passes `chainId` as `eip155:${hexChainId}`)

4.1.4 V-KLS-VUL-004: Query injection pitfalls for future link feature

Severity	Info	**Commit**	34d1332
Type	Query Injection	**Status**	Fixed
File(s)	`index.ts`		
Location(s)	getInsights()		
Fixed At	ca8253d		

Several comments indicate that the developers intend to implement a "deeplink" feature when MetaMask Snaps supports Markdown links. However, the example links seem to use JavaScript template strings, which means that the interpolated parameters will be directly inserted into the links without any escaping. When the developers implement the features mentioned in the

```
1  else {
2    // Contract was not tagged in Address Tags. Let the user know, and provide a link
       to tag it.
3    // Note: current @metamask/snaps-ui does not allow markdown links, so no links in
       this version.
4    // todo: when links are a feature, turn them into [Tag me](https://curate.kleros.
     io/...), deeplink:
5    // https://curate.kleros.io/tcr/100/0x66260c69d03837016d88c9877e61e08ef74c59f2?
     action=submit&Public%20Name%20Tag=&Contract%20Address=${contractAddress}
6    const addressNotFound = '**Contract Tag:** _Not Found_';
7    insights.push(addressNotFound);
8  }
9  const domainLabel = result.contractDomain
10   ? '**Domain:** _${domain}_ is **verified** for this contract'
11   : // todo: when links are a feature, deeplink:
12     // https://curate.kleros.io/tcr/100/0x957A53A994860BE4750810131d9c876b2f52d6E1?
     action=submit&Contract%20Address=${caipAddress}&Domain%20Name=${domain}
13     '**Domain:** _${domain}_ is **NOT verified** for this contract
```

Snippet 4.4: Location of the comments in `getInsights()`

comments, we recommend that the developers use safe URL construction APIs like the `URL` class, `URLSearchParams`, and/or `encodeURIComponent()` . This can help reduce the attack surface and prevent query injection attacks.

Developer Response The developers do not see this issue as exploitable as the address and domain will be supplied by the MetaMask execution environment, so that they will likely not contain characters such as `?` or `&`. However, the developers acknowledge that it would be good practice to use the safe URL construction APIs.

4.1.5 V-KLS-VUL-005: Some values fetched by query are unused

Severity	Info	**Commit**	34d1332
Type	Maintainability	**Status**	Fixed
File(s)	`index.ts`		
Location(s)	fetchGraphQLData()		
Fixed At	48e6481		

There are some values which are fetched by the GraphQL query, but they are not used anywhere in the program.

- `itemID` is fetched in all fields of the query, but it is not used to construct any of the `CuratedInfo` fields.
- `key4` is fetched for `addressTags`, but it is not used to construct the `AddressTag`.

```
const parsedAddressTag: AddressTag | undefined = result.data.addressTags[0]
? {
    caipAddress: mdEscape(result.data.addressTags[0].key0),
    publicName: mdEscape(result.data.addressTags[0].key1),
    projectName: mdEscape(result.data.addressTags[0].key2),
    infoLink: mdEscape(result.data.addressTags[0].key3),
  }
: undefined;
```

Snippet 4.5: Location where the `addressTag` values are used.

Developer Response The developers agree that the values mentioned above are unused and will remove them.

4.1.6 V-KLS-VUL-006: Lack of error handling when sending GraphQL query

Severity	Info	**Commit**	34d1332
Type	Logic Error	**Status**	Fixed
File(s)	`index.ts`		
Location(s)	fetchGraphQLData()		
Fixed At	6a03929, 7489a72		

The `fetchGraphQLData()` function will use the fetch API to make an HTTP request. This can fail in the following situations, but there is no logic that handles these failures:

- There is a `NetworkError` when making the HTTP request in `fetchGraphQLData()`. Note that there is no try-catch logic to handle this error type in `fetchGraphQLData()` or its callers.
- `fetchGraphQLData()` does not check the status code of the HTTP response (e.g., with `request.ok` or `request.status`).

The developers may want to add logic to handle the errors and show a user-friendly error message if such errors occur.