Veridise. Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Onthis MetaMask Snap

Veridise Inc.

March 6, 2024

► **Prepared For:**

Onthis
https://www.onthis.xyz/

► **Prepared By:**

Benjamin Mariano
Jacob Van Geffen

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

► **Version History:**

Mar. 06 2024 V2
Feb. 09, 2024 V1

Contents

1 Executive Summary

From Feb. 8, 2024 to Feb. 9, 2024, Onthis engaged Veridise to review the security of their Onthis MetaMask Snap. The review covered the client's Metamask snap implementation, which is used to notify users on transactions with verification that the transaction recipient is a valid Onthis Shortcut contract. The snap also provides an estimate on rewards points that will be earned by valid transactions. Veridise conducted the assessment over 4 person-days, with 2 engineers reviewing code over 2 days on commit `6a8c7ed`. The auditing strategy involved extensive manual auditing performed by Veridise engineers.

Code assessment. The Onthis MetaMask Snap developers provided the source code of the Onthis MetaMask Snap contracts for review. The code is based off of MetaMask's example snap infrastructure and mostly consists of very basic API calls. Documentation of the code itself is scarce, although it is fairly straightforward. Developers did provide documentation for their more general Shortcut infrastructure as well as a video showing some of the intended behavior of the snap.

As far as Veridise auditors can tell, the source code contains no tests.

Summary of issues detected. The audit uncovered 4 issues, including one critical severity issue (V-OTH-VUL-001). The critical issue is an incorrectly specified chain ID which means no verification is ever performed. In addition to this error, there were a few other low, warning, and info-level errors, including issues with imprecise user-facing messages (V-OTH-VUL-002) and some other usability issues.

Recommendations. After auditing the protocol, in addition to fixing the bugs shared in the report, the auditors suggest that the developers should consider adding tests of basic functionality for the snap. The most major issue likely could have been avoided via basic testing.

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
Onthis MetaMask Snap	`6a8c7ed`	TypeScript	MetaMask Snap

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Feb. 8 - Feb. 9, 2024	Manual	2	4 person-days

Table 2.3: Vulnerability Summary.

Name	Number	Fixed	Acknowledged
Critical-Severity Issues	1	1	1
High-Severity Issues	0	0	0
Medium-Severity Issues	0	0	0
Low-Severity Issues	1	1	1
Warning-Severity Issues	0	0	0
Informational-Severity Issues	2	2	2
TOTAL	4	4	4

Table 2.4: Category Breakdown.

Name	Number
Logic Error	1
Usability Issue	1
Maintainability	1
Optimization	1

3 Audit Goals and Scope

3.1 Audit Goals

The engagement was scoped to provide a security assessment of Onthis MetaMask Snap's MetaMask snap.

- Are all transactions appropriately verified to be sent to Onthis's Shortcut contracts?
- Does estimation of rewards follow the intended formula?
- Does the protocol leave users vulnerable to phising attacks – e.g., do user prompts clearly describe actions?
- Will the snap operate only on Mainnet transactions as intended?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions above, our audit involved intense manual auditing by human experts. Auditors consulted relevant documentation of the snap itself as well as more general documentation including MetaMask snap, Supabase, and more general Onthis Shortcuts documentation.

Scope. The scope of this audit is limited to the code in the `/packages/snap` folder of the source code provided by the Onthis MetaMask Snap developers, which contains the snap implementation.

Methodology. Veridise auditors first read the Onthis MetaMask Snap documentation as well as related documentation for the surrounding tooling (MetaMask, TypeScript, Supabase, etc.). They then began a manual audit of the code. During the audit, the Veridise communicated issues and asked questions with the Onthis MetaMask Snap developers via a Telegram chat.

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

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-OTH-VUL-001	Incorrect chain ID	Critical	Fixed
V-OTH-VUL-002	Imprecise user message	Low	Fixed
V-OTH-VUL-003	Type mismatch in function return	Info	Fixed
V-OTH-VUL-004	Check for Mainnet chain ID should be moved up	Info	Fixed

4.1 Detailed Description of Issues

4.1.1 V-OTH-VUL-001: Incorrect chain ID

Severity	Critical	**Commit**	6a8c7ed
Type	Logic Error	**Status**	Fixed
File(s)	`index.ts`		
Location(s)	onTransaction()		
Confirmed Fix 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. As per the CAIP-2 standard, the chain ID has the format `namespace:id` where `namespace` covers a class of blockchains and `id` is an integer identifier for a particular chain.

In the snap, the developers only want to validate chains on Mainnet and thus include the following check:

```
if(chainId !== '1') {
    return {
      content: panel([
        divider(),
        text(‘Shortcuts availabe ONLY‘),
        text(‘for MAINNET addresses‘),
        divider(),
      ]),
    };
  }
```

Snippet 4.1: Snippet from `onTransaction()`

However, their check `chainId !== '1'` will actually be satisfied by all chain IDs in the CAIP-2 standard, meaning all transactions (including Mainnet transactions) will be denied.

Impact No transactions will be validated, even Mainnet ones which are intended to be validated.

Recommendation Change the check to use the proper CAIP-2 ID for Mainnet: `eip155:1`.

Developer Response Shortcut validations has been changed to `if(chainId !== 'eip155:1')` as suggested.

4.1.2 V-OTH-VUL-002: Imprecise user message

Severity	Low	**Commit**	6a8c7ed
Type	Usability Issue	**Status**	Fixed
File(s)	`index.ts`		
Location(s)	onTransaction()		
Confirmed Fix At	N/A		

The main function of this snap is to validate that the recipient of a transaction matches one of Onthis's Shortcut contracts. This validation is done by fetching all of the Shortcuts and checking that one matches the transaction's `to` address field. Presuming this check is successful and a Shortcut contract is matched, an estimate of the reward points earned is computed and the following message is shown to the user indicating the transaction has been verified:

```
content: panel([
  text('Shortcut: ${validatedShortcutData.ens_name}'),
  divider(),
  text('Contract: ${validatedShortcutData.address}'),
  divider(),
  text('Verified by ONTHIS '),
  divider(),
  text('Est. Points Receive: ${estimatedPoints}')
]),
```

Snippet 4.2: Message after Shortcut verified

A potential issue is that the message indicates the user that the transaction is verified but the verification had nothing to do with the actual transaction data and no information about the transaction data is presented to the user.

Impact This may lead to a user falling victim to a scam where they accidentally verify a transaction (believing it has been verified by Onthis) because the scam made the user believe they are sending a transaction which they are not.

For instance, suppose the user believes they are transferring 1 ETH using the Shortcut toArbitrum.eth from their account on Ethereum to their account on Arbitrum. However, in reality, a malicious actor has tricked them into sending 1 ETH from their account on Ethereum to *the attacker's* account on Arbitrum. The malicious transaction will be "verified" by the Snap and all information presented to the user would be identical to the transaction the user intends to send.

Recommendation Add additional information to the verification message indicating other details about the transaction. Or, at the very least, add some disclaimers that verification only means the shortcut address is correct, and does not indicate any verification of the validity of the transaction itself.

Developer Response `validatedShortcutData.address` address variable has been changed to `transaction.to` which always will show the proper "to" address. Additionally, we now display

the `validatedShortcutData.description` which gives more detail about the transaction being executed.

```
return {
      content: panel([
        text(‘Shortcut: ${validatedShortcutData.ens_name}‘),
        divider(),
        text(‘Shortcut contract: ${transaction.to}‘),
        divider(),
        text(
          ‘ ${
            validatedShortcutData.desciption
              ? ‘ Shortcut Description:‘ + validatedShortcutData.desciption
              : ’ ’
          }‘,
        ),
        divider(),
        text(‘Verified by ONTHIS    ‘),
        divider(),
        text(‘Est. Points Receive: ${estimatedPoints}‘)
      ]),
    };
```

4.1.3 V-OTH-VUL-003: Type mismatch in function return

Severity	Info	**Commit**	6a8c7ed
Type	Maintainability	**Status**	Fixed
File(s)	`helper/index.ts`		
Location(s)	estimateRewardPoints()		
Confirmed Fix At	N/A		

The function `estimateRewardPoints()` returns the estimated reward points earned by the user for the validated transaction which is computed as described here. After it is computed, the reward points amount are rounded to an integer using the `toFixed` call (or just `0` is returned in the case that no rewards are earned) as shown below:

```
return stage.length
  ? (value * validatedShortcutData.complexity * stage[0].stage_multiplier / 10**19).
    toFixed(0)
  : 0;
```

Snippet 4.3: Snippet from `estimateRewardPoints()`

The concern here is that `toFixed` returns a `string` while `0` is an `integer`.

Impact At the moment, the code should work fine in both cases, as the caller just adds the value to the message sent to the user, which will convert the value to a string as necessary. However, future iterations of the code might miss this difference and make incorrect assumptions about the behavior of the code.

Recommendation Standardize the output type of the function to be either an `integer` or a `string`.

Developer Response Added `Promise<string>` as return type and return value has been standardized to `string` as well.

```
export const estimateRewardPoints = async (
  value: any,
  validatedShortcutData: any,
  supabase: any,
):Promise<string>  => {
  const { data: stage } = await supabase
    .from('points_distribution_state')
    .select('*');

  return stage.length
    ? (value * validatedShortcutData.complexity * stage[0].stage_multiplier / 10**19)
    .toFixed(0)
    : "0";
};
```


4.1.4 V-OTH-VUL-004: Check for Mainnet chain ID should be moved up

Severity	Info	**Commit**	6a8c7ed
Type	Optimization	**Status**	Fixed
File(s)	`index.ts`		
Location(s)	onTransaction()		
Confirmed Fix At	N/A		

The Onthis snap only supports Mainnet transactions, and so checks the `chainId` of the transaction. Before checking, the snap performs several queries to validate the shortcut that was used.

```
const supabase = await createSupabaseClient();
const { data: shortcuts } = await supabase.from('shortcuts').select('*');
const validatedShortcutData = validateShortcut(
  shortcuts,
  transaction.to as string,
);

if(chainId !== '1') {
  // return an error panel
}
```

Snippet 4.4: Snippet from `onTransaction()`

Since the result of these queries is not used when `chainId` does not correspond to Mainnet, the check should be moved up to occur before the queries.

Impact If `chainId` does not correspond to Mainnet, the snap will issue server queries that will not be used.

Recommendation Move the check against `chainId` to the start of `onTransaction` to avoid unnecessary server queries.

Developer Response Code was optimized to avoid unnecessary calls to the database by moving validation to the top of function body as was suggested.

```
export const onTransaction: OnTransactionHandler = async ({ transaction, chainId })
    => {
  if(chainId !== 'eip155:1') {
    return {
      content: panel([
        divider(),
        text('Shortcuts availabe ONLY'),
        text('for MAINNET addresses'),
        divider(),
      ]),
    };
  }
```