Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Pentest-Report Hedera Wallet Snap & Sources 04.2024

Cure53, Dr.-Ing. M. Heiderich, B. Casaje

Index

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Introduction

“Hedera Wallet Snap unlocks wallet functionality for Hedera via MetaMask that any other apps can interact with, thereby turning MetaMask into a native Hedera wallet without relying on Hedera JSON-RPC Relay.”

From https://docs.tuum.tech/hedera-wallet-snap

This report describes the result of a penetration test and source code audit against the Hedera Wallet Snap, as well as its codebase, performed by Cure53 in Q2 2024.

To give some context regarding the assignment’s origination and composition, Tuum Technologies Inc. contacted Cure53 in March 2024. The test execution was scheduled for CW15 April 2024, whereby four work days were invested to reach the coverage expected for this project. A team of two senior testers was assigned to this project’s preparation, execution and finalization.

The work was structured using a single work package (WP), defined as:

- **WP1**: Pen.-tests & code audits against Hedera Wallet Snap & codebase

Note that this was not the first time the Hedera Wallet Snap and codebase were audited by Cure53, as they were already the focus of an audit held in November 2023 (see TUU-02).

The methodology conformed to a white-box strategy, whereby assistive materials such as sources, test-supporting documentation, as well as all further means of access required to complete the tests were provided to facilitate the undertakings.

All preparations were completed in early April 2024, specifically during CW14, to ensure a smooth start for Cure53. Communication throughout the test was conducted through a dedicated and shared Slack channel, established to combine the teams of Tuum and Cure53.

All personnel involved from both parties were invited to participate in this channel. Communications were smooth, with few questions requiring clarification, and the scope was well-defined and clear. No significant roadblocks were encountered during the test. Cure53 provided frequent status updates and shared their findings, and offered live reporting through the aforementioned Slack channel.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

The Cure53 team achieved good coverage over the scope items, and identified a total of four findings. Of the four security-related discoveries, two were classified as security vulnerabilities, and two were categorized as general weaknesses with lower exploitation potential.

Cure53's audit of the Hedera Wallet Snap revealed a well-secured environment. The platform demonstrates strong defenses against common Snap vulnerabilities, and left a very positive impression on the testing team regarding its security posture.

The team did not identify any issues of *Critical* severity during this audit - which speaks positively of the platform's security. However, one finding needs to be highlighted due to its severity. This finding addresses a scenario in which it is possible to switch the mirror node without requiring any confirmation from the user (see TUU-03-002), and was therefore ranked with a *High* severity.

Although the Hedera Wallet Snap's security posture can already be considered quite robust, in order to further strengthen this, Cure53 advises that all of the findings and issues detailed in this report should be swiftly actioned.

The report will now shed more light on the scope and testing setup, as well as provide a comprehensive breakdown of the available materials. Next, the report will detail the *Methodology* used in this exercise. This chapter will show which areas of the software in scope have been covered and what tests have been executed, despite the limited number of findings made during the course of the exercise. Following this, the report will list all findings identified in chronological order, starting with the *Identified Vulnerabilities* and followed by the *Miscellaneous Issues* unearthed. Each finding will be accompanied by a technical description, Proof-of-Concepts (PoCs) where applicable, plus any fix or preventative advice to action.

In summation, the report will finalize with a *Conclusions* chapter in which the Cure53 team will elaborate on the impressions gained toward the general security posture of the Hedera Wallet Snap, as well as its codebase.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Scope

- **Code audits & security reviews against Hedera Wallet Snap & related codebase**
  - **WP1**: Pen.-tests & code audits against Hedera Wallet Snap & codebase
    - **Sources:**
      - https://github.com/hashgraph/hedera-metamask-snaps/tree/main
    - **Commit:**
      - 2d164945740a2615ad9eeb6e5ea6c234ac18c873
    - **Primary focus:**
      - https://github.com/hashgraph/hedera-metamask-snaps/tree/2d164945740a2615ad9eeb6e5ea6c234ac18c873/packages/hedera-wallet-snap/packages/snap
    - **Documentation:**
      - https://docs.tuum.tech/hedera-wallet-snap
  - **Test-supporting material was shared with Cure53**
  - **All relevant sources were shared with Cure53**

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Testing Methodology

This section outlines the testing methodology and coverage achieved during the engagement, shedding light on various components of the Hedera Wallet MetaMask Snap and codebase that Cure53 inspected. Further clarification is given for the areas of investigation that were subject to deep-dive assessment, while the test team also specifies the techniques applied to evaluate the respective security posture of each component.

This scope of the audit was solely the Hedera Wallet Snap, and not the site for testing the Snap. This approach complied with a white-box pentesting strategy, as the full source code for the Snap was open-source.

The test started off with a review of the scope, provided documentation, and the Manifest file of the Snap. This was done to generally verify if the Snap requested any unused endowment or permissions. No unused permissions were found. Then, the audit team reviewed the code of the Snap to see if it adhered with web security best practices, and checked if any hardening guidance related to MetaMask Snaps could be applied.

Next, a focus was placed on the usage of the *endowment:network-access* permission. This endowment enables the Snap to make network requests, which could potentially leak sensitive information to third-parties, for example, through logging. However, all places using *fetch* in the Snap only request the mirror node URL. The testing team found no evidence that sensitive information was being transferred to third-party services.

Each RPC method was then carefully reviewed for prevalent access control and injection weaknesses. Cure53 noted that communication is performed by webpages and dApps via MetaMask's *wallet_invokeSnap* request, which guarantees that the aforementioned application possesses all necessary privileges before permitting interaction with the Snap.

The team found an injection issue where user input in multiple places was directly fed into the *text* method, allowing for markdown and control characters to be rendered in various dialogs. This would allow a malicious dApp to spoof some elements of the UI, potentially phishing the user (TUU-03-001).

The team also investigated whether any of the RPC methods were vulnerable to type confusion, missed bounds checks, or failed to sanitize data before use. All of the RPC methods explicitly check whether *request.params* is valid before use. This includes checking that all of the required parameters exist, and that they are of the correct type and format. This implementation was found to be very secure, and helped to greatly reduce the attack surface of the Snap.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

While looking for injection vulnerabilities, the team noticed that there were multiple places vulnerable to client-side path traversal. There were multiple instances of user-input being directly placed into the URL for a network request, without any escaping. As such, the user input could contain characters to change the path or add query parameters (TUU-03-004).

Next, the team checked to see if all actions taken by the Snap were intuitive, clearly telegraphed, and required clear user consent. Almost all of the RPC methods were found to use the *snap_dialog* method provided by MetaMask to get confirmation from the user. However, one issue was discovered where the *network* and *mirrorNodeURL* parameters could change the network used by the Snap without any user consent (TUU-03-002).

The team checked all methods to ensure that all sensitive data was handled correctly. No methods were found that disclosed or logged sensitive information like private keys or the Snap state. However, some methods implemented error handling incorrectly, returning a caught unsanitized error object which contained full stack traces (TUU-03-003).

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Identified Vulnerabilities

The following section lists all vulnerabilities and implementation issues identified during the testing period. Notably, findings are cited in chronological order rather than by degree of impact, with the severity rank offered in brackets following the title heading for each vulnerability. Furthermore, all tickets are given a unique identifier (e.g., TUU-03-001) to facilitate any future follow-up correspondence.

TUU-03-001 WP1: Markdown and control characters allowed in dialogs *(Medium)*

***Fix Note**: This issue was fixed and the fix was verified by Cure53 in early April 2024. The documented problem no longer exists.*

Cure53 observed that some of the dialogs shown in the Snap display user input, using the *text* method provided by the MetaMask UI. However, this process is problematic since the *text* method permits rendering control characters and Markdown. Notably, both newlines and bold text can be injected into these fields.

A malicious dApp is granted a number of potential input control methods that would be rendered via the text method, such as *signMessage*, *hts/createToken*, or *hts/updateToken*. These could spoof different fields in the UI, potentially forcing a user into accidentally signing a message or confirming a transaction.

In addition, the *signMessage* method takes an optional *header* parameter, which is not included in the signature, and can be used to display any arbitrary text.

PoC:

```
await ethereum.request({
    method: 'wallet_invokeSnap',
    params: {snapId:`local:http://localhost:9001`,
        request:{
            method: 'signMessage',
            params:{
               header: "**NOTE**: Transaction has been verified to not be
                    malicious.",
               message: "**The following is the message to be signed:**\n
                    \nsign data\n\n**Spoofed field**: x"
            }
        }
    }
})
```

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

To mitigate this issue, Cure53 advises enforcing that user input is never passed into the *text* method. Alternatively, utilize the *copyable* method to display text as recommended by MetaMask[1], which ignores Markdown and other special characters. In addition, remove the *header* parameter from the *signMessage* method if possible.

TUU-03-002 WP1: Mirror node switchable without user confirmation *(High)*

***Fix Note**: This issue was fixed and the fix was verified by Cure53 in early April 2024. The documented problem no longer exists.*

All Snap methods take the optional *mirrorNodeUrl* and *network* parameters which allows for custom Hedera mirror nodes to be used. For example, these parameters can be set to use Testnet, Mainnet, or any other custom network. However, this feature is implemented insecurely, as none of the dialogs shown to the user display the network for the operation.

This could be used by a malicious dApp to phish a user. The UI could display that the user is connected to Testnet or any non-Mainnet node, and methods like *getAccountInfo* could even run, showing the user's Testnet balance to confuse them further. Then, the malicious dApp could run a more sensitive method, like *transferCrypto*, but instead use the Mainnet mirror node behind the scenes, which the user would not expect. This could lead to loss of user funds.

In addition, a malicious dApp could also set a network which proxies traffic between the Snap and the real mirror nodes, getting access to potentially sensitive data like signatures. However, due to time constraints and the fact that the functionality of the mirror nodes is out of scope for this audit, this attack avenue was not explored further.

To fix this issue, it is recommended to make switching the network a Snap method that requires user confirmation to ensure that the user is always aware of which mirror node they are connected to. Alternatively, it is recommended to make the selected network visible at the top of all relevant confirmations.

[1] https://docs.metamask.io/snaps/learn/best-practices/security-guidelines/

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Miscellaneous Issues

This section covers any and all noteworthy findings that did not incur an exploit but may assist an attacker in successfully achieving malicious objectives in the future. Most of these results are vulnerable code snippets that did not provide an easy method by which to be called. Conclusively, while a vulnerability is present, an exploit may not always be possible.

TUU-03-003 WP1: Some methods return full stack traces upon error *(Info)*

***Fix Note**: This issue was fixed and the fix was verified by Cure53 in early April 2024. The documented problem no longer exists.*

It was discovered that multiple Snap methods do not handle errors correctly, returning full stack traces upon error. This is undesirable, because error stacks could potentially contain sensitive information, such as the *SnapState*, and a malicious dApp can catch these messages and read their contents.

The implementation of multiple methods use *try* and *catch* to catch errors, but then re-throw these same errors in the *catch* block, leaking error stacks and messages. However, the team was not able to find a Snap method that would leak an error stack containing sensitive information due to the audit's time-constrained nature, so this finding was deemed to be a miscellaneous issue.

PoC:

```
try {
    await ethereum.request({
        method: 'wallet_invokeSnap',
        params: {snapId:`local:http://localhost:9001`,
            request:{
                method: 'transferCrypto',
                params:{
                    transfers: [{
                        assetType: 'HBAR',
                        to: '0.0.123',
                        amount: 1
                    }]
                }
            }
        }
    })
}
catch (err) {
    console.log("caught error:", err)
}
```

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Steps to reproduce:

1. Install Metamask and the Hedera Wallet Snap.
2. Connect the Snap to a wallet that has an account on the Hedera network.
3. Open the console in DevTools, and run the PoC above.
4. A dialog confirmation box should appear. Click *Reject*.
5. The stack trace should be caught, and should appear in the console log.

To fix this issue, it is recommended to check all Snap methods and ensure they only throw custom error messages that do not provide stack traces or debugging information.

TUU-03-004 WP1: Client-side path traversal in various methods *(Info)*

Fix Note: This issue was fixed and the fix was verified by Cure53 in early April 2024. The documented problem no longer exists.

It was discovered that some Snap methods place user input directly into the path and query parameters without any escaping. This allows for client-side path traversal, since if user input contained *../*, it would change the endpoint being requested by the Snap. In places where user input is used as a query parameter unescaped, other parameters could be injected with an ampersand.

However, it was noted that the URLs for the requests are intended to be public mirror nodes that can be switched out for other URLs by the dApp directly. Given that the team was not able to exploit this issue, this was not deemed a vulnerability.

PoC:

```
await ethereum.request({
    method: 'wallet_invokeSnap',
    params: {snapId:`local:http://localhost:9001`,
        request:{
            method: 'hts/deleteToken',
            params:{
                tokenId: '../../xxx',
            }
        }
    }
})
```

To fix this issue, it is recommended to encode all user input that is placed into the path or query parameters with a function like *encodeURIComponent*.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Conclusions

As noted in the *Introduction*, the Hedera Wallet Snap left a very robust impression on the Cure53 team during this April 2024 penetration test in terms of its security posture. This is supported by the limited number of issues outlined in this report, and the lack of any vulnerabilities of *Critical* severity being found.

From a contextual perspective, four working days were allocated to reach the coverage expected for this project. A white-box penetration testing methodology was used to assess the specified scope of the application. This approach included direct examination of the underlying source code for the Snap. The assessment confirmed that the Hedera team had successfully avoided and mitigated a number of typical web application and Snap risks.

To facilitate productive communication and seamless information exchange between the Hedera and Cure53 teams, a dedicated Slack channel was set up. In addition to this, the clearly defined scope of the assessment and a shared understanding of target areas minimized any potential roadblocks throughout the assessment process. The Cure53 team was provided with documentation for the Snap as well as a video detailing new features to help facilitate testing.

The testing team effectively covered a significant portion of the scoped area during their audit. However, Cure53 must emphasize that the modest list of defects detailed in this report is likely due to the limited functionality that existed within the scope, as well as the minimal attack surface that exists for Snaps in general.

The code in the *snap* folder was reviewed extensively for any web vulnerabilities and common Snap misconfigurations.

During testing, the first finding noted by the Cure53 team was that user input was being fed directly into the *text* method provided by MetaMask for dialogs, which allows markdown and control characters to be rendered (TUU-03-001). In addition, the *signMessage* RPC method was found to take a *header* parameter which could inject additional fields, and additionally would not be a part of the signature. This could lead to UI spoofing attacks, where an attacker could inject fields and phish the user.

A *High* severity issue was found where dApps using the Snap could provide the *network* and *mirrorNodeURL* parameters when calling RPC method, which would change the network used by the Snap without any user consent (TUU-03-002). This could easily confuse the user as to which mirror node and network they were interacting with, and in the worst case, lead to direct loss of user funds.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

The Cure53 team discovered that some of the RPC methods return full stack traces upon error. Most of the RPC method implementations use *try* and *catch* to handle errors that occur during execution. Some methods insecurely return the caught error directly, which gives the dApp using the Snap access to this error directly. This error is not sanitized, which means it contains the full stack trace and could potentially contain sensitive information, but a way to exploit this was not found. For more information, see ticket TUU-03-003.

Some methods were also found to place user input directly into a URL that is then fetched without any escaping or sanitization. This leads to client-side path traversal, as user input could contain characters that change the path or query parameters (TUU-03-004). The team was unable to find a potential attack scenario for this issue.

In conclusion, Cure53's penetration test of the Hedera Wallet Snap paints a picture of a well-secured environment, with strong defenses against a variety of common web vulnerabilities. Cure53 garnered a unanimously strong impression regarding the security offering established by the Hedera Wallet Snap during this engagement. Fixing the vulnerabilities found should be relatively trivial and will further strengthen the environment's security posture.

Cure53 would like to thank Kiran Pachhai and Donald Bullers from the Tuum Technologies, Inc. team for their excellent project coordination, support and assistance, both before and during this assignment.