Metamask Snap Audit Report for Avail

Testers:

1. Or Duan
2. Avigdor Sason Cohen

Table of Contents

Management Summary

Avail contacted Sayfer Security in order to perform penetration testing on Avail's MetaMask Snap in 04/2024.

Before assessing the above services, we held a kickoff meeting with the Avail technical team and received an overview of the system and the goals for this research.

Over the research period of 2 weeks, we discovered 6 vulnerabilities in the system.

In conclusion, several fixes should be implemented following the report, but the system's security posture is competent.

Risk Methodology

At Sayfer, we are committed to delivering the highest quality penetration testing to our clients. That's why we have implemented a comprehensive risk assessment model to evaluate the severity of our findings and provide our clients with the best possible recommendations for mitigation.

Our risk assessment model is based on two key factors: **IMPACT** and **LIKELIHOOD**. Impact refers to the potential harm that could result from an issue, such as financial loss, reputational damage, or a non-operational system. Likelihood refers to the probability that an issue will occur, taking into account factors such as the complexity of the attack and the number of potential attackers.

By combining these two factors, we can create a comprehensive understanding of the risk posed by a particular issue and provide our clients with a clear and actionable assessment of the severity of the issue. This approach allows us to prioritize our recommendations and ensure that our clients receive the best possible advice on how to protect their business.

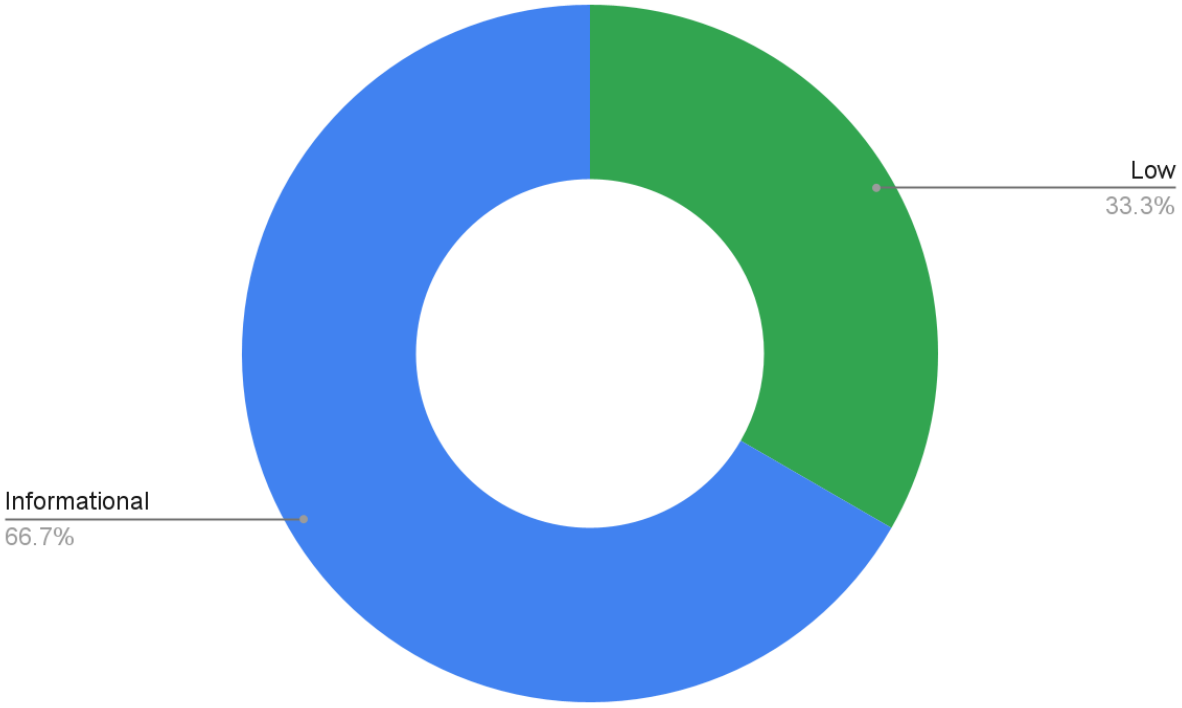
Risk is defined as follows:

Overall Risk Security

IMPACT ➤				
	HIGH	Medium	High	High
	MEDIUM	Low	Medium	High
	LOW	Informational	Low	Medium
		LOW	MEDIUM	HIGH

LIKELIHOOD ➤

Vulnerabilities by Risk



Risk	Low	Medium	High	Informational
# of issues	2	0	0	4

- **Low** – No direct threat exists. The vulnerability may be exploited using other vulnerabilities.
- **Medium** – Indirect threat to key business processes or partial threat to business processes.
- **High** – Direct threat to key business processes.
- **Informational** – This finding does not indicate vulnerability, but states a comment that notifies about design flaws and improper implementation that might cause a problem in the long run.

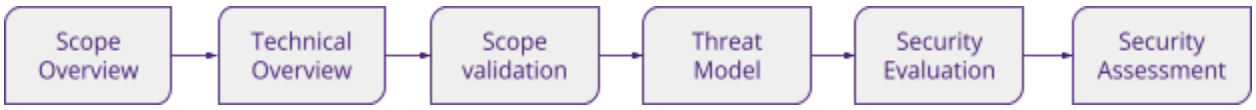
Approach

Introduction

Avail contacted Sayfer to perform penetration testing on their MetaMask Snap application.

This report documents the research carried out by Sayfer targeting the selected resources defined under the research scope. Particularly, this report displays the security posture review for Avail's MetaMask Snap application and its surrounding infrastructure and process implementations.

Our penetration testing project life cycle:



Scope Overview

During our first meeting and after understanding the company's needs, we defined the application's scope that resides at the following URLs as the scope of the project:

- Avail's MetaMask Snap
  - **Audit commit:** be57a78
  - **Fixes commit:**

Our tests were performed from 25/04/2024 to 09/05/2024.

Scope Validation

We began by ensuring that the scope defined to us by the client was technically logical.
Deciding what scope is right for a given system is part of the initial discussion. Getting the scope right is key to deriving maximum business value from the research.

Threat Model

During our kickoff meetings with the client we defined the most important assets the application possesses.
We defined that the largest current threat to the system is leakage of sensitive user information.

Security Evaluation Methodology

Sayfer uses OWASP WSTG as our technical standard when reviewing web applications. After gaining a thorough understanding of the system we decided which OWASP tests are required to evaluate the system.

Security Assessment

After understanding and defining the scope, performing threat modeling, and evaluating the correct tests required in order to fully check the application for security flaws, we performed our security assessment.

Issue Table Description

Issue title

ID	**SAY-??**: An ID for easy communication on each vulnerability
Status	Open/Fixed/Acknowledged
Risk	Represents the risk factor of the issue. For further description refer to the Vulnerabilities by Risk section.
Business Impact	The main risk of the vulnerability at a business level.
Location	The URL or the file in which this issue was detected. Issues with no location have no particular location and refer to the product as a whole.
Description	Here we provide a brief description of the issue and how it formed, the steps we made to find or exploit it, along with proof of concept (if present), and how this issue can affect the product or its users.
Mitigation	Suggested resolving options for this issue and links to advised sites for further remediation.

Security Evaluation

The following tests were conducted while auditing the system

Information Gathering	Test Name
WSTG-INFO-01	Conduct Search Engine Discovery Reconnaissance for Information Leakage
WSTG-INFO-02	Fingerprint Web Server
WSTG-INFO-03	Review Webserver Metafiles for Information Leakage
WSTG-INFO-04	Enumerate Applications on Webserver
WSTG-INFO-05	Review Webpage Content for Information Leakage
WSTG-INFO-06	Identify application entry points
WSTG-INFO-07	Map execution paths through application
WSTG-INFO-08	Fingerprint Web Application Framework
WSTG-INFO-09	Fingerprint Web Application
WSTG-INFO-10	Map Application Architecture

Configuration and Deploy Management Testing	Test Name
WSTG-CONF-01	Test Network Infrastructure Configuration
WSTG-CONF-02	Test Application Platform Configuration
WSTG-CONF-03	Test File Extensions Handling for Sensitive Information
WSTG-CONF-04	Review Old Backup and Unreferenced Files for Sensitive Information
WSTG-CONF-05	Enumerate Infrastructure and Application Admin Interfaces
WSTG-CONF-06	Test HTTP Methods
WSTG-CONF-07	Test HTTP Strict Transport Security
WSTG-CONF-08	Test RIA cross domain policy
WSTG-CONF-09	Test File Permission
WSTG-CONF-10	Test for Subdomain Takeover
WSTG-CONF-11	Test Cloud Storage

Identity Management Testing	Test Name
WSTG-IDNT-01	Test Role Definitions
WSTG-IDNT-02	Test User Registration Process
WSTG-IDNT-03	Test Account Provisioning Process

WSTG-IDNT-04	Testing for Account Enumeration and Guessable User Account
WSTG-IDNT-05	Testing for Weak or unenforced username policy

Authentication Testing	Test Name
WSTG-ATHN-01	Testing for Credentials Transported over an Encrypted Channel
WSTG-ATHN-02	Testing for Default Credentials
WSTG-ATHN-03	Testing for Weak Lock Out Mechanism
WSTG-ATHN-04	Testing for Bypassing Authentication Schema
WSTG-ATHN-05	Testing for Vulnerable Remember Password
WSTG-ATHN-06	Testing for Browser Cache Weaknesses
WSTG-ATHN-07	Testing for Weak Password Policy
WSTG-ATHN-08	Testing for Weak Security Question Answer
WSTG-ATHN-09	Testing for Weak Password Change or Reset Functionalities
WSTG-ATHN-10	Testing for Weaker Authentication in Alternative Channel

Authorization Testing	Test Name
WSTG-ATHZ-01	Testing Directory Traversal File Include
WSTG-ATHZ-02	Testing for Bypassing Authorization Schema
WSTG-ATHZ-03	Testing for Privilege Escalation
WSTG-ATHZ-04	Testing for Insecure Direct Object References

Session Management Testing	Test Name
WSTG-SESS-01	Testing for Session Management Schema
WSTG-SESS-02	Testing for Cookies Attributes
WSTG-SESS-03	Testing for Session Fixation
WSTG-SESS-04	Testing for Exposed Session Variables
WSTG-SESS-05	Testing for Cross Site Request Forgery
WSTG-SESS-06	Testing for Logout Functionality
WSTG-SESS-07	Testing Session Timeout
WSTG-SESS-08	Testing for Session Puzzling
WSTG-SESS-09	Testing for Session Hijacking

Data Validation Testing	Test Name
WSTG-INPV-01	Testing for Reflected Cross Site Scripting

WSTG-INPV-02	Testing for Stored Cross Site Scripting
WSTG-INPV-03	Testing for HTTP Verb Tampering
WSTG-INPV-04	Testing for HTTP Parameter Pollution
WSTG-INPV-05	Testing for SQL Injection
WSTG-INPV-06	Testing for LDAP Injection
WSTG-INPV-07	Testing for XML Injection
WSTG-INPV-08	Testing for SSI Injection
WSTG-INPV-09	Testing for XPath Injection
WSTG-INPV-10	Testing for IMAP SMTP Injection
WSTG-INPV-11	Testing for Code Injection
WSTG-INPV-12	Testing for Command Injection
WSTG-INPV-13	Testing for Format String Injection
WSTG-INPV-14	Testing for Incubated Vulnerability
WSTG-INPV-15	Testing for HTTP Splitting Smuggling
WSTG-INPV-16	Testing for HTTP Incoming Requests
WSTG-INPV-17	Testing for Host Header Injection
WSTG-INPV-18	Testing for Server-side Template Injection
WSTG-INPV-19	Testing for Server-Side Request Forgery

Error Handling	Test Name
WSTG-ERRH-01	Testing for Improper Error Handling
WSTG-ERRH-02	Testing for Stack Traces

Cryptography	Test Name
WSTG-CRYP-01	Testing for Weak Transport Layer Security
WSTG-CRYP-02	Testing for Padding Oracle
WSTG-CRYP-03	Testing for Sensitive Information Sent via Unencrypted Channels
WSTG-CRYP-04	Testing for Weak Encryption

Business logic Testing	Test Name
WSTG-BUSL-01	Test Business Logic Data Validation
WSTG-BUSL-02	Test Ability to Forge Requests
WSTG-BUSL-03	Test Integrity Checks
WSTG-BUSL-04	Test for Process Timing
WSTG-BUSL-05	Test Number of Times a Function Can be Used Limits
WSTG-BUSL-06	Testing for the Circumvention of Work Flows
WSTG-BUSL-07	Test Defenses Against Application Mis-use

WSTG-BUSL-08	Test Upload of Unexpected File Types
WSTG-BUSL-09	Test Upload of Malicious Files

Client Side Testing	Test Name
WSTG-CLNT-01	Testing for DOM-Based Cross Site Scripting
WSTG-CLNT-02	Testing for JavaScript Execution
WSTG-CLNT-03	Testing for HTML Injection
WSTG-CLNT-04	Testing for Client Side URL Redirect
WSTG-CLNT-05	Testing for CSS Injection
WSTG-CLNT-06	Testing for Client Side Resource Manipulation
WSTG-CLNT-07	Test Cross Origin Resource Sharing
WSTG-CLNT-08	Testing for Cross Site Flashing
WSTG-CLNT-09	Testing for Clickjacking
WSTG-CLNT-10	Testing WebSockets
WSTG-CLNT-11	Test Web Messaging
WSTG-CLNT-12	Testing Browser Storage
WSTG-CLNT-13	Testing for Cross Site Script Inclusion

API Testing	Test Name
WSTG-APIT-01	Testing GraphQL

Security Assessment Findings

Logic Error during API Reset

ID	SAY-01
Status	Open
Risk	Low
Business Impact	Incomplete API reset, which may lead to unforeseen consequences.
Location	- …/src/avail/api.ts:34; resetApi()
Description	During the configure method, if the API has an initial configuration, i.e. `state.config` is not null, an API reset is performed via the `resetAPI()` function, which validates whether the variables `api` and `provider` exist and If they do, sets them to null. But if only one parameter is null, the reset will not take place and the second parameter will remain in an unexpected state, which may lead to incorrect assumptions and further issues down the line. `export const resetApi = async (): Promise<void> => { if (api && provider) { try { await api.disconnect(); } catch (e) { console.error('Error on api disconnect.'); } api = null; provider = null; } };`
Mitigation	Change the AND (&) operator to an OR (|) in the if statement. If either of the values exists, both should be reset.

Inconsistent Error Handling

ID	SAY-02
Status	Open
Risk	Low
Business Impact	Function execution will not be halted, which may lead to unexpected results.
Location	- …/src/avail/api.ts:38; resetApi()
Description	If the parameters `api` and `provider` exist during API reset, `api.disconnect()` operation is executed in the try-catch block. If it fails and "catch" catches an error, it is displayed in the console. However, the error isn't "thrown" any further. This means that despite its occurrence, code execution continues and the API and provider are reset. This may be problematic if the API has not been successfully disconnected and some network traffic is still occurring. ``` export const resetApi = async (): Promise<void> => { if (api && provider) { try { await api.disconnect(); } catch (e) { console.error('Error on api disconnect.'); } api = null; provider = null; } }; ``` For comparison, `initApi()` also uses the try-catch mechanism, however, if an error is detected, it is thrown using the throw statement, interrupting execution.
Mitigation	Review the snap's logic and decide whether `resetApi()` should throw an error and disrupt execution if `api.disconnect()` fails.

Misleading Error Message

ID	SAY-03
Status	Open
Risk	Informational
Business Impact	The returned error may be confusing.
Location	- …/src/index.ts:91; onRpcRequest()
Description	It has been noticed that several of the `onRpdRequests` implement schema-based validation. The parameters passed when calling a message must match the expected format and types. Otherwise, an error is returned. While most schemes have generic error messages, `validConfigureSchema` implements a custom text: "Network name should be provided". This is unhelpful because this scheme has many more parameters, such as `wsRpcUrl` or `addressPrefix`. If any of them are incorrect, the error returned will not be true. `// set new configuration` `assert(` `  request.params,` `  validConfigureSchema,` `  'Invalid configuration schema - Network name should be provided'` `);`
Mitigation	We suggest either adapting the error message to be more generic, or verifying which parameter caused the error and returning specific, detailed information to the caller.

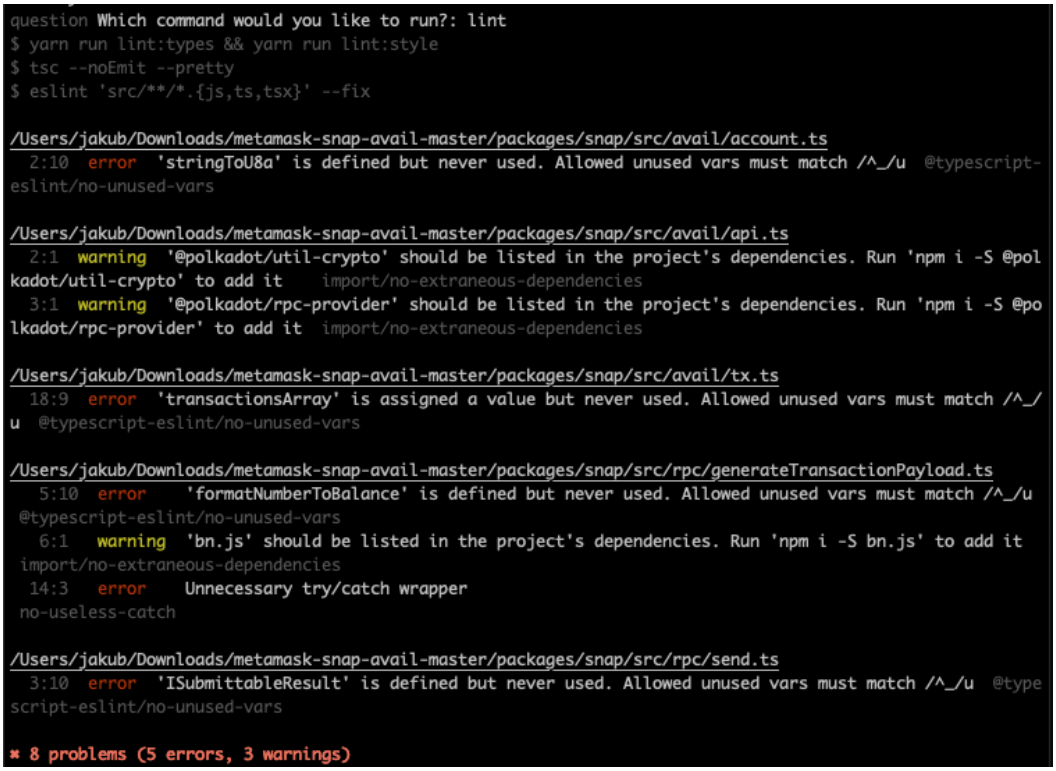
Missing Return

ID	SAY-04
Status	Open
Risk	Informational
Business Impact	Lack of adherence to common programming standards.
Location	- …/src/rpc/substrate/sign.ts - signPayloadJSON(ApiPromise,SignerPayloadJSON) - signPayloadRaw(ApiPromise,SignerPayloadJSON)
Description	`signPayloadJSON` and `signPayloadRaw`, called via the corresponding `onRpcRequest` handlers, return `Promise<{ signature: string } | void>` if the confirmation dialog displayed to the user is. Then the execution is successful. However, if the user has not accepted the prompt, the function ends without performing any further action, returning an implicit undefined. In this context, a better solution would be to return null or an error message to the caller.
Mitigation	To improve readability, consider always returning a clearly defined value in each branch of a function's execution. For example, you can modify the function to always return either a signed object, explicitly null, or a special error object when the user does not confirm the operation. This approach increases readability and facilitates debugging.

Redundant Code Fragments

ID	SAY-05
Status	Open
Risk	Informational
Business Impact	Reducing the amount of redundant code will improve code readability and performance.
Location	- …/src/avail/api.ts:53; getApi() - …/src/configuration/index.ts; getDefaultConfiguration(string) - …/src/avail/accounts.ts; getCoinTypeByNetwork(SnapNetworks) - …/src/avail/tx.ts; updateTxInState(Transaction)
Description	Some code fragments seem redundant or duplicated. This reduces the readability of the code, makes debugging more difficult and unnecessarily increases the number of lines that require analysis. For instance: • `getApi()` calls `initApi(string)`. At the end of its logic, `initApi(string)` displays the string "Api is ready" in the JavaScript console and returns an api object. Then, `getApi()` logs the exact same object, only now with the string "API". This seems unnecessary because we are logging the same data twice. • `getDefaultConfiguration(string)` is supposed to return the network configuration. However, since there is currently only one configuration available, the default Avail configuration, the whole function seems a bit pointless. Until you support another program, this piece of code could be safely removed. • In a similar vein, `getCoinTypeByNetwork(SnapNetworks)` is supposed to return different coin types depending on the network indicated in the parameter. However, there is currently only one network type and therefore coin type – avail – so the returned value will always be 354. • `updateTxInState(Transaction)` is not used by any of the handlers implemented in the snap and can therefore be removed.
Mitigation	We recommend analyzing the indicated fragments and possibly cleaning up unnecessary code.

Linter Not Used

ID	SAY-06
Status	Open
Risk	Informational
Business Impact	While there are no direct security benefits for making sure your codebase is formatted consistently, it does enhance the snap's readability and professionality.
Location	—
Description	It seems like a linter was not consistently used to analyze the code. This leads to situations where some files have unused imports and variables, and redundant try-catch blocks. • Example of some of the errors caught by running a linter: 
Mitigation	A quick and painless fix would be to run a linter, such as `yarn lint`, over the code base.

```
question Which command would you like to run?: lint
$ yarn run lint:types && yarn run lint:style
$ tsc --noEmit --pretty
$ eslint 'src/**/*.{js,ts,tsx}' --fix

/Users/jakub/Downloads/metamask-snap-avail-master/packages/snap/src/avail/account.ts
  2:10  error  'stringToU8a' is defined but never used. Allowed unused vars must match /^_/u  @typescript-
eslint/no-unused-vars

/Users/jakub/Downloads/metamask-snap-avail-master/packages/snap/src/avail/api.ts
  2:1  warning  '@polkadot/util-crypto' should be listed in the project's dependencies. Run 'npm i -S @pol
kadot/util-crypto' to add it    import/no-extraneous-dependencies
  3:1  warning  '@polkadot/rpc-provider' should be listed in the project's dependencies. Run 'npm i -S @po
lkadot/rpc-provider' to add it  import/no-extraneous-dependencies

/Users/jakub/Downloads/metamask-snap-avail-master/packages/snap/src/avail/tx.ts
  18:9  error  'transactionsArray' is assigned a value but never used. Allowed unused vars must match /^_/
u  @typescript-eslint/no-unused-vars

/Users/jakub/Downloads/metamask-snap-avail-master/packages/snap/src/rpc/generateTransactionPayload.ts
   5:10  error    'formatNumberToBalance' is defined but never used. Allowed unused vars must match /^_/u
 @typescript-eslint/no-unused-vars
   6:1   warning  'bn.js' should be listed in the project's dependencies. Run 'npm i -S bn.js' to add it
 import/no-extraneous-dependencies
  14:3   error    Unnecessary try/catch wrapper
 no-useless-catch

/Users/jakub/Downloads/metamask-snap-avail-master/packages/snap/src/rpc/send.ts
  3:10  error  'ISubmittableResult' is defined but never used. Allowed unused vars must match /^_/u  @type
script-eslint/no-unused-vars

✖ 8 problems (5 errors, 3 warnings)
```

We are available at security@sayfer.io

If you want to encrypt your message please use our public PGP key:

https://sayfer.io/pgp.asc

Key ID: 9DC858229FC7DD38854AE2D88D81803C0EBFCD88

Website: https://sayfer.io

Public email: info@sayfer.io

Phone: +972-559139416