Metamask Snap Audit Report for AlephZero

Testers:

1. Or Duan
2. Omri Shdaimah

Table of Contents

Table of Contents **2**
Management Summary **3**
Risk Methodology **4**
Vulnerabilities by Risk **5**
Approach **6**
Introduction 6
Scope Overview 7
Scope Validation 7
Threat Model 7
Security Evaluation Methodology 8
Security Assessment 8
Issue Table Description 9
Security Evaluation **10**
Security Assessment Findings **14**
Lack of Validation in uint8ArrayFromHex(string) 14
Type Checking Is not Uniform 15
Unused Functions 16
Dependency with Outstanding Vulnerability 17
Lack of Documentation and Commenting 18

Management Summary

AlephZero contacted Sayfer Security in order to perform penetration testing on AlephZero's MetaMask Snap in November 2023.

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

Over the research period of 2 weeks, 3 vulnerabilities were found in the system. Also, some suggestions were added to the report for ongoing improvements.

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

After a review by the Sayfer team, we certify that all the security issues mentioned in this report have been addressed by the AlephZero team.

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.

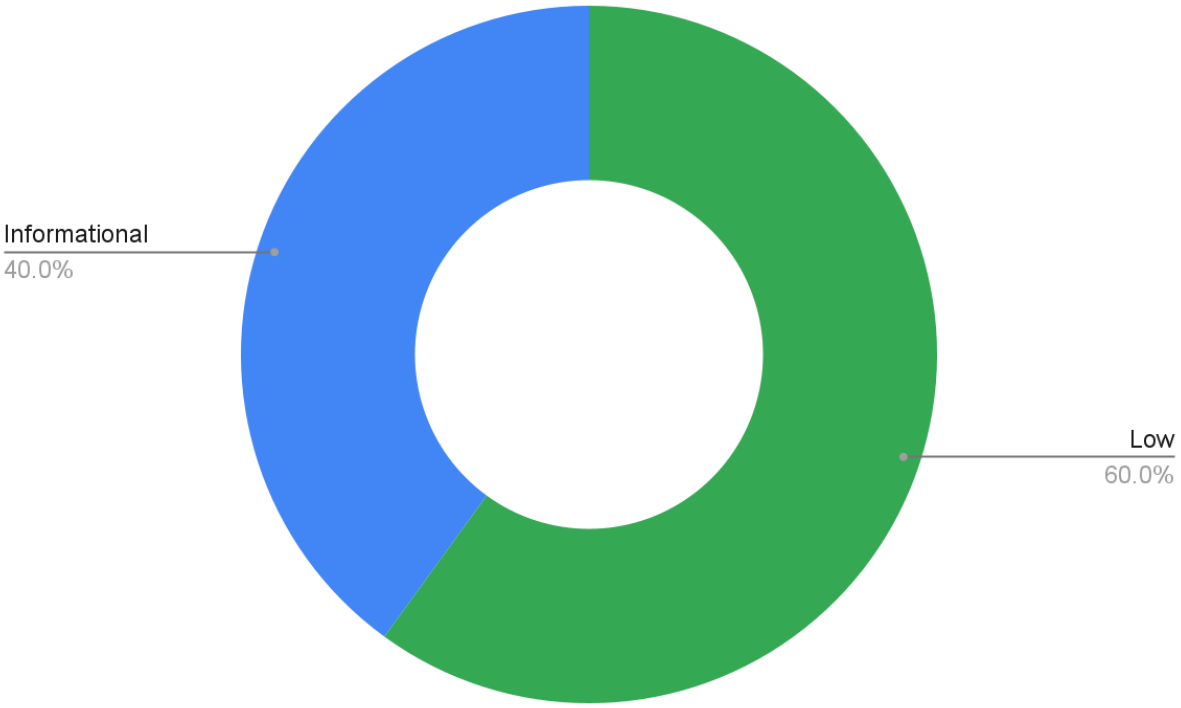
By combining these two factors, we can create a comprehensive understanding of the risk posed by a particular issue and provide our clients with a clear and actionable assessment of the severity of the issue. This approach allows us to prioritize our recommendations and ensure that our clients receive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

Risk is defined as follows:

Overall Risk Security

IMPACT ↓ / LIKELIHOOD →	LOW	MEDIUM	HIGH
HIGH	Medium	High	High
MEDIUM	Low	Medium	High
LOW	Informational	Low	Medium

Vulnerabilities by Risk



Risk	Low	Medium	High	Informational
# of issues	3	0	0	2

- **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.

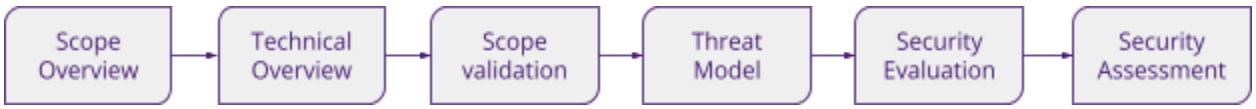
Approach

Introduction

AlephZero 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 AlephZero'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:

- AlephZero's MetaMask Snap
  - **Audit commit**: 5e6aa033a94c69be7b2b82e875a20c69c5dd5274
  - **Fixes commit:** f8105a7322c13aed378fe71eaf3d4d5cf7736869

Our tests were performed from November to December 2023.

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 the potential for malicious attackers to steal funds from other users.

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

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

Identity Management Testing	Test Name	Status
WSTG-IDNT-01	Test Role Definitions	Pass

WSTG-IDNT-02	Test User Registration Process	Pass
WSTG-IDNT-03	Test Account Provisioning Process	Pass
WSTG-IDNT-04	Testing for Account Enumeration and Guessable User Account	Pass
WSTG-IDNT-05	Testing for Weak or unenforced username policy	Pass

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

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

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

Data Validation Testing	Test Name	Status
WSTG-INPV-01	Testing for Reflected Cross Site Scripting	Pass
WSTG-INPV-02	Testing for Stored Cross Site Scripting	Pass
WSTG-INPV-03	Testing for HTTP Verb Tampering	Pass
WSTG-INPV-04	Testing for HTTP Parameter Pollution	Pass
WSTG-INPV-05	Testing for SQL Injection	Pass
WSTG-INPV-06	Testing for LDAP Injection	Pass
WSTG-INPV-07	Testing for XML Injection	Pass
WSTG-INPV-08	Testing for SSI Injection	Pass
WSTG-INPV-09	Testing for XPath Injection	Pass
WSTG-INPV-10	Testing for IMAP SMTP Injection	Pass
WSTG-INPV-11	Testing for Code Injection	Pass
WSTG-INPV-12	Testing for Command Injection	Pass
WSTG-INPV-13	Testing for Format String Injection	Pass
WSTG-INPV-14	Testing for Incubated Vulnerability	Pass
WSTG-INPV-15	Testing for HTTP Splitting Smuggling	Pass
WSTG-INPV-16	Testing for HTTP Incoming Requests	Pass
WSTG-INPV-17	Testing for Host Header Injection	Pass
WSTG-INPV-18	Testing for Server-side Template Injection	Pass
WSTG-INPV-19	Testing for Server-Side Request Forgery	Pass

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

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

Business logic Testing	Test Name	Status
WSTG-BUSL-01	Test Business Logic Data Validation	Pass
WSTG-BUSL-02	Test Ability to Forge Requests	Pass

WSTG-BUSL-03	Test Integrity Checks	Pass
WSTG-BUSL-04	Test for Process Timing	Pass
WSTG-BUSL-05	Test Number of Times a Function Can be Used Limits	Pass
WSTG-BUSL-06	Testing for the Circumvention of Work Flows	Pass
WSTG-BUSL-07	Test Defenses Against Application Mis-use	Pass
WSTG-BUSL-08	Test Upload of Unexpected File Types	Pass
WSTG-BUSL-09	Test Upload of Malicious Files	Pass

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

API Testing	Test Name	Status
WSTG-APIT-01	Testing GraphQL	Pass

Security Assessment Findings

Lack of Validation in *uint8ArrayFromHex(string)*

ID	SAY-01
Status	Fixed
Risk	Low
Business Impact	Rather than raising an error when a wrong input is supplied, the function will simply attempt the conversion and return a wrong result.
Location	- packages/snap/src/utils.ts:15; uint8ArrayFromHex(string)
Description	It has been found that *uint8ArrayFromHex(string)* does not validate whether the input *hexString*, which is of type string, is actually a hexadecimal. • uint8ArrayFromHex(string): `export const uint8ArrayFromHex = (hexString: string) => { const strBytes = hexString.replace(/^0x/iu, '').match(/../gu) ?? []; return new Uint8Array(strBytes.map((byte: string) => parseInt(byte, 16))) .buffer; };` The function only replaces the "0x" characters with an empty string, as well as converting bytes to integers. Then, the buffer is returned as a *uint8* string. Consequently, if such validation is not performed before calling the function, an incorrect string may be processed.
Mitigation	Validate that the function's input is in fact a hexadecimal.

Type Checking Is not Uniform

ID	SAY-02
Status	Fixed
Risk	Low
Business Impact	Using the *any* type, the snap loses control over the initial type of the variable, which may have consequences in the dependent logic that implements it.
Location	- `packages/snap/src/index.ts:21; onRpcRequest(string, JsonRpcRequest)` - `packages/snap/src/services/storage.ts:6`
Description	*any* type casting was sometimes used throughout the snap. This is considered a bad security practice, since it deprives us of important type information and may consequently lead to unexpected runtime errors when unexpected types are used.
Mitigation	When unsure about a type, they should be defined down using "unknown", and then narrowed down by type guards.

Unused Functions

ID	SAY-03
Status	Fixed
Risk	Low
Business Impact	Unused functions may imply that some logic has not yet been created or has been forgotten, as well as adding unnecessary code volume.
Location	- `packages/snap/src/utils.ts; getRandomBytes(number), uint8ArrayFromHex(string), sha256(string)`
Description	While examining the code, we noticed that none of the functions in `utils.ts` are implemented anywhere in the codebase. The specified functions all have content, but are never called, making their presence in the code seemingly unnecessary. This may be due to future code development, scrapping of certain functionalities, or a bug.
Mitigation	Make sure that the specified functions are necessary, and if they are, make sure to use them.

Dependency with Outstanding Vulnerability

ID	SAY-04
Status	Fixed
Risk	Informational
Business Impact	The relevant package is a third-order subdependency of `@metamask/snaps-cli@1.0.2`, so upgrading it is not your responsibility and it's hard to assess the exact impact. But nevertheless, it is important to be aware of such risks, and to keep your direct dependencies always on the latest version. Therefore, we decided to include this finding as informational.
Location	—
Description	By running `pnpm audit`, we found out that one subdependency of your project, `browserify-sign@4.2.1`, is associated with a high risk vulnerability that may be relevant to your project.
Mitigation	Update `@metamask/snaps-cli@1.0.2` as soon as possible and periodically run `pnpm audit`.

high	**browserify-sign upper bound check issue in `dsaVerify` leads to a signature forgery attack**
Package	browserify-sign
Vulnerable versions	>=2.6.0 <=4.2.1
Patched versions	>=4.2.2
Paths	packages/snap > @metamask/snaps-cli@1.0.2 > browserify@17.0.0 > crypto-browserify@3.12.0 > browserify-sign@4.2.1
More info	https://github.com/advisories/GHSA-x9w5-v3q2-3rhw

Lack of Documentation and Commenting

ID	SAY-05
Status	Acknowledged
Risk	Informational
Business Impact	The lack of proper documentation and commenting may be perceived as unprofessional by some users. Users may also be weary about installing a snap whose exact functionality they do not quite understand.
Location	—
Description	The README file and several generic articles are the only materials describing the snap and its functionalities. Additionally, these articles do not match the code and function names that are implemented.
Mitigation	Write a proper readme and make sure to comment the code.