Metamask Snap Audit Report for Openverse

Testers:

1. Or Duan
2. Avigdor Sason Cohen

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**
Origin Spoofing 14
Non-standard Signing Implementation 15
Missing Transaction Broadcasting Functionality 16
Missing Anti-Replay Protection 17
Missing Input Sanitization for User-Facing Content 18
Confusing Dialog Box 19

Management Summary

Openverse contacted Sayfer Security in order to perform penetration testing on Openverse's MetaMask Snap, Openverse Wallet, in 02/2025.

Before assessing the above services, we held a kickoff meeting with the Openverse 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.

After review by the Sayfer team, we certify that the high-risk vulnerability mentioned in this report has been fixed and that all others have been acknowledged by the Openverse 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.

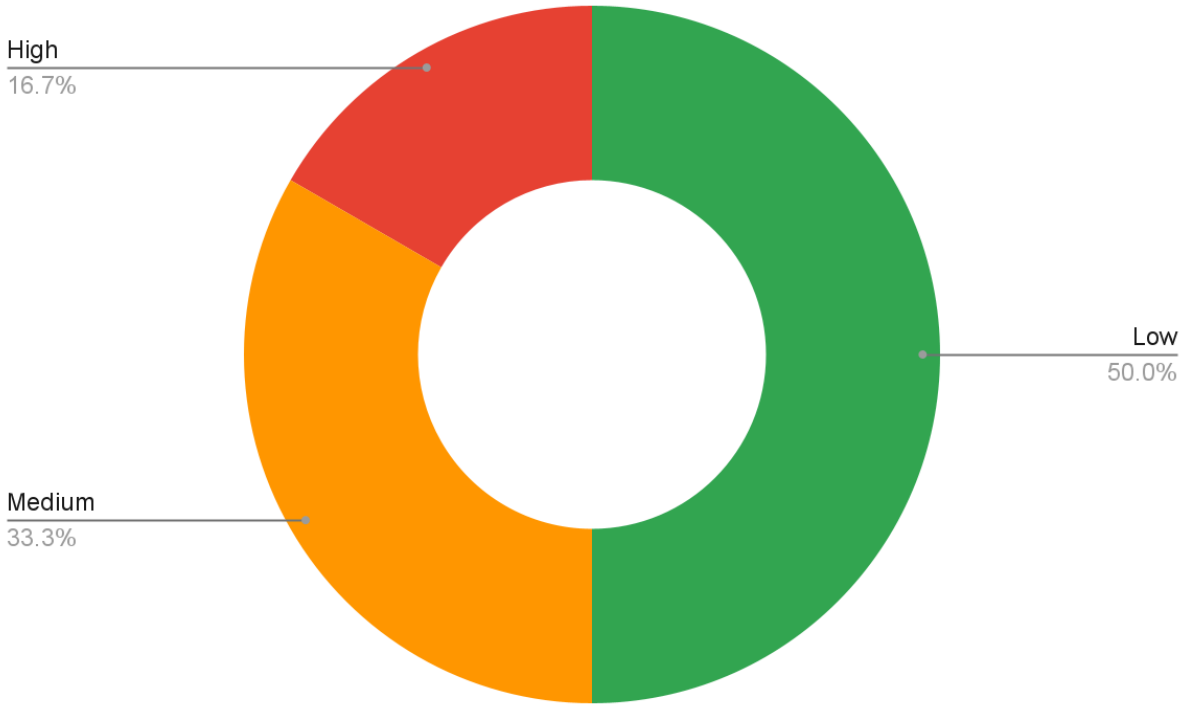
Risk is defined as follows:

Overall Risk Security

IMPACT ❯				
	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	3	2	1	0

- **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.

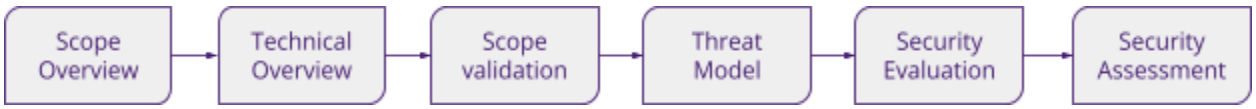
Approach

Introduction

Openverse contacted Sayfer to perform penetration testing on their MetaMask Snap application, Openverse Wallet.

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 Openverse Wallet 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:

- Openverse Wallet
  - **Audit commit:** `6517c60c29399823ccb41dbcbd3cd2602762baaa`
  - **Fixes commit:** `7eec0ae321506b92e7f949a18917bcec360ba6f7`

Our tests were performed from 10/02/2025 to 18/02/2025.

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 the largest current threat to the system as phishing attacks.

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

Origin Spoofing

ID	SAY-01
Status	Fixed
Risk	High
Business Impact	User controlled origin may allow malicious dapps to impersonate trusted domains in confirmation dialogs, leading to phishing attacks.
Location	- `index.tsx; onRpcRequest()`
Description	The snap prioritizes the origin value provided in *`request.params`* (controlled by the calling dapp) over the validated *`origin`* parameter provided by MetaMask. This allows attackers to spoof their displayed origin in UI dialogs (e.g., showing *`openverse.network`* while the real origin is *`phishing.site`*). MetaMask's same-origin policy ensures the origin parameter reflects the true invoking domain, but the snap ignores this safeguard. • `index.tsx:22` `const dappOrigin = (request?.params as { origin?: string }).origin || origin;`
Mitigation	We recommend using MetaMask's validated origin parameter.

Non-standard Signing Implementation

ID	SAY-02
Status	Acknowledged
Risk	Medium
Business Impact	The usage of a homebrewed signing implementation bypasses MetaMask's security model and thereby needlessly increases the risk of key leakage.

Location	- index.tsx; onRpcRequest(any, any)
	  - case signTransaction
	  - case signAllTransactions
	  - case signMessage

Description	The snap uses *tweetnacl* to sign raw messages with directly derived private keys (*secretKey*), instead of leveraging MetaMask's native methods, such as *eth_signTypedData_v4*. This violates MetaMask's key isolation principles and potentially introduces novel vulnerabilities.
	• index.tsx:62; case signTransaction
	`const signature = nacl.sign.detached(bs58.decode(message), keyPair.secretKey);`
	• index.tsx:81-84; case signAllTransactions
	`const signatures = messages .map((message: string) => bs58.decode(message)) .map((message: Uint8Array) => nacl.sign.detached(message, keyPair.secretKey)) .map((signature: Uint8Array | number[]) => bs58.encode(signature));`
	• index.tsx:113; case signMessage
	`const signature = nacl.sign.detached(messageBytes, keyPair.secretKey);`

Mitigation	We recommend replacing the custom signing logic with *eth_signTypedData_v4*.

Missing Transaction Broadcasting Functionality

ID	SAY-03
Status	Acknowledged
Risk	Medium
Business Impact	Signatures are returned to untrusted dapps, which could misuse them.

Location	- index.tsx; onRpcRequest(any, any)
	  - case signTransaction
	  - case signAllTransactions
	  - case signMessage

Description	The snap signs transactions and messages but does not broadcast them, relying entirely on the dapp to handle on-chain submission. This increases reliance on potentially malicious dapps.
	• index.tsx:64-67, 115-118; case signTransaction, case signMessage
	`return { publicKey: bs58.encode(keyPair.publicKey), signature: bs58.encode(signature) };`
	• index.tsx:86-89; case signAllTransactions
	`return { publicKey: bs58.encode(keyPair.publicKey), signatures };`

Mitigation	We recommend implementing transaction broadcasting, or alternatively informing the user that their transaction will be handled by the dapp.

Missing Anti-Replay Protection

ID	SAY-04
Status	Acknowledged
Risk	Low
Business Impact	Signatures can be replayed across chains or contexts, potentially leading to unintended asset transfers.

Location	- index.tsx; onRpcRequest(any, any)
	  - case signTransaction
	  - case signAllTransactions
	  - case signMessage

Description	The snap signs raw messages without including chain-specific identifiers such as chainId, domain separators (e.g., EIP-712 domain), or nonces. Attackers could reuse signatures on other networks where the same message is valid.
Mitigation	We recommend prepending chain-specific prefixes such as `\x19Openverse:\n{chainId}` to messages before signing.

Missing Input Sanitization for User-Facing Content

ID	SAY-05
Status	Acknowledged
Risk	Low
Business Impact	The lack of input sanitization increases the risk of phishing or UI spoofing via maliciously crafted messages containing Markdown or deceptive links.
Location	- `index.tsx; renderSignMessage(string, string)`

Description	`renderSignMessage(string, string)` displays raw user-provided messages using the `<Text>` component, which interprets Markdown syntax. An attacker could inject hyperlinks such as `[Legit Site](evil.site)` to trick users into approving malicious content.
	• `index.tsx:202-209`
	`content: (`
	`  <Box>`
	`  <Heading>Sign message</Heading>`
	`  <Text>{host}</Text>`
	`  <Divider />`
	`  <Text>{message}</Text>`
	`  </Box>`
	`)`

Mitigation	Replace `<Text>` with the `<Copyable>` component for displaying messages to prevent Markdown rendering and ensure content is treated as plaintext.

Confusing Dialog Box

ID	SAY-06
Status	Acknowledged
Risk	Low
Business Impact	Users may accidentally approve malicious transactions due to information overload or lack of individual review.
Location	- `index.tsx; renderSignAllTransactions(string, any)`
Description	`renderSignAllTransactions(string, any)` displays multiple transactions in a single dialog, making it difficult for users to scrutinize each transaction. This increases the likelihood of approving unintended or harmful actions. • `index.tsx:202-209`

```
for (let i = 0; i < messages.length; i++) {
 uiElements.push(<Divider />);
 // uiElements.push(Text(`Transaction ${i + 1}`));
 uiElements.push(<Text>Transaction {(i + 1).toString()}</Text>);
 // uiElements.push(Copyable(messages[i]));
 uiElements.push(<Copyable value={messages[i]}></Copyable>);
}

return snap.request({
 method: 'snap_dialog',
 params: {
   type: 'confirmation',
   content:
     (
       <Box>
         <Heading>Sign transactions</Heading>
         <Text>{host}</Text>
         {uiElements}
       </Box>
     )
                                  [ ... ]
}
```

Mitigation	Split bulk transactions into individual confirmation dialogs or add a summary screen highlighting critical details, for example total value or list of recipients before approval.

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