

Metamask Snap Audit Report for Coti

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**
No Confirmation Dialogue in set-aes-key 14
No Confirmation Dialogue in import-token and hide-token 15
get-aes-key Returns Raw AES Keys to Any Dapp 16
Arbitrary Digest Signing without Meaningful Context 17
SSRF via NFT Metadata 18
onUpdate Clears State without Warning or Migration 19
Weak Address Validation in ERC20 Import Flow 20
Unvalidated JSON.parse on Untrusted Input 21

Management Summary

Coti contacted Sayfer Security in order to perform penetration testing on Coti's MetaMask Snap in 03/2026.

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

Over the research period of 2 weeks, we discovered 7 vulnerabilities in the system.

In conclusion, 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 Coti 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	4	0	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.

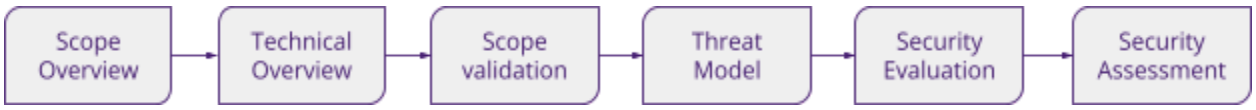
Approach

Introduction

Coti 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 Coti'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:

- Coti's MetaMask Snap
  - **Audit commit:** ac5d55a1edd011db0c214d2f83bb0d811e35305f
  - **Fixes commit:** 39ac8e58a4238401371ca798f4371732b8ec5c18

Our tests were performed from 12/03/2026 to 26/03/2026.

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 the user's AES encryption key, which protects confidential balances and underpins wallet key derivation, leading to potential loss of funds and full compromise of confidential transaction data.

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

No Confirmation Dialogue in `set-aes-key`

ID	SAY-01
Status	Fixed
Risk	Medium
Business Impact	A malicious or compromised dapp with RPC access can silently replace the user's encryption key with an attacker-controlled one, causing all subsequent confidential token transfers to be encrypted with the attacker's key. The attacker can then decrypt all confidential balances and transaction amounts, while the user loses the ability to view their own balances.
Location	- `.../src/index.tsx:817`
Description	The `set-aes-key` RPC handler directly writes the provided `newUserAesKey` to state without any user-facing confirmation dialog. This stands in contrast with other sensitive RPC handlers in the same file: `delete-aes-key` requires a full `snap_dialog` confirmation and `get-aes-key` also prompts the user for approval before returning key material. Additionally, the snap manifest declares `"endowment:rpc": { "dapps": true }`, meaning any dapp origin can invoke this method, i.e. there is no whitelist of trusted origins. The handler also accepts any non-empty string as a valid AES key without checking hex format or length, making it possible to set weak or malformed keys.
Mitigation	• Add a `snap_dialog` confirmation before writing the AES key, clearly showing the requesting origin and a warning about key replacement. • Validate the AES key format (must be exactly 32 hex characters / 16 bytes for AES-128) before storing. • Consider restricting the allowed origins for this sensitive operation to only the companion dapp or alternatively to a whitelist of approved dapps.

No Confirmation Dialogue in `import-token` and `hide-token`

ID	SAY-02
Status	Fixed
Risk	Medium
Business Impact	Similarly to SAY-01, any dapp can silently add or remove tokens from the user's snap token list without user consent. A malicious dapp can flood the list with phishing tokens that mimic legitimate assets to trick users into interacting with malicious contracts, or silently remove legitimate tokens from the user's view.
Location	- `.../src/index.tsx:945, 1006`
Description	The `import-token` handler accepts arbitrary token addresses, names, symbols, and decimals from any calling dapp and writes them directly to the snap's persistent state with no user confirmation dialog. Similarly, `hide-token` removes tokens by address without any confirmation. Both handlers completely bypass the user. This is inconsistent with the snap's own internal UI flow for importing ERC20/ERC721 tokens via `onUserInput`, which requires user interaction through form validation and UI confirmation steps. Since the snap manifest grants RPC access to all dapps, any website can invoke these methods.
Mitigation	Add `snap_dialog` confirmation for both `import-token` and `hide-token` operations, displaying the token details (address, name, symbol) and the requesting origin before modifying the token list.

`get-aes-key` Returns Raw AES Keys to Any Dapp

ID	SAY-03
Status	Fixed
Risk	Medium
Business Impact	While the handler does prompt for user approval, the confirmation dialog uses vague wording that does not communicate that raw cryptographic key material will be sent to the requesting website. Once a dapp obtains the raw AES key, it can decrypt all confidential balances and craft encrypted transfer amounts offline.
Location	- `.../src/index.tsx:724`
Description	`get-aes-key` returns the raw value of `currentAESState.aesKey` to the calling dapp after the user approves a dialog. The confirmation message reads “Approve to unlock your security key”, but does not explicitly warn that the raw cryptographic key will be exposed to the website. Users may approve this dialog thinking it is a benign authentication or unlock operation, not realizing they are handing over the master encryption key that protects all their confidential balances. The generic wording provides inadequate informed consent for such a sensitive operation. Any dapp can trigger this dialog. There is no restriction on which origins may request the AES key, meaning phishing sites or compromised dapps can obtain it if the user clicks "Approve".
Mitigation	Rewrite the confirmation dialog to be explicit – “This will share your raw AES encryption key with {origin}. This key can decrypt all your confidential balances and transactions.”

Arbitrary Digest Signing without Meaningful Context

ID	SAY-04
Status	Fixed
Risk	Medium
Business Impact	Any dapp can request the snap to sign an arbitrary 32-byte digest using the snap-derived private key. The confirmation dialog only shows raw hex, which is uninterpretable to users. This could be exploited to forge signatures for COTI MPC precompile operations or other on-chain actions tied to the snap's derived wallet address.
Location	- `.../src/index.tsx:656`
Description	The `sign-raw-256` handler signs any 32-byte hex value using `buildRawSignature()`, which derives a wallet from `deriveSnapWallet(aesKey, address, chainId)`. The only validation performed is `isHex32()`, a check that the input is exactly 32 bytes of hex data. The user sees a dialog displaying the raw hex digest, which is completely opaque to human interpretation. Consequently, they have no way of knowing whether they are signing a benign operation or authorizing a malicious transaction. This is analogous to the well-known `eth_sign` blind-signing vulnerability in MetaMask, which MetaMask itself deprecated for exactly this reason.
Mitigation	• Add structured contextual data to the signing dialog explaining what is being signed. • Consider restricting the handler to known function selectors or requiring additional context parameters.

SSRF via NFT Metadata

ID	SAY-05
Status	Fixed
Risk	Low
Business Impact	A malicious token contract can return a crafted `tokenURI` pointing to internal network addresses or cloud metadata endpoints, allowing server-side request forgery from the MetaMask extension context.
Location	- `... /src/utils/image.ts:54`
Description	The snap requests the `endowment:network-access` permission and fetches URLs derived from on-chain NFT token URI metadata. In `fetchWithGatewayFallback`, non-IPFS URLs are fetched directly via `fetchWithTimeout(string, ?number)` with no URL scheme validation, host validation, or IP address filtering. The URI value originates from decrypted on-chain token metadata (`getTokenURI( ... )` in token.ts), which is fully attacker-controlled if they deploy a malicious NFT contract and trick the user into importing it. The attacker can set the `tokenURI` to point to internal network resources like `http://169.254.169.254/` for cloud metadata, or `http://localhost:PORT/` for local services. `getSVGfromMetadata(string)` then fetches the image field from the returned JSON response, creating a second SSRF opportunity - the attacker controls both the initial URI and the URI within the metadata JSON.
Mitigation	Validate that fetched URLs use only `https://` or `ipfs://` schemes. Block private IP ranges like `10.0.0.0/8`, `172.16.0.0/12`, `192.168.0.0/16` and link-local addresses like `169.254.0.0/16`.

Weak Address Validation in ERC20 Import Flow

ID	SAY-06
Status	Fixed
Risk	Low
Business Impact	The snap's internal ERC20 import flow validates token addresses by string length only, not by Ethereum address format. Any 42-character string passes validation, potentially leading to confusing error messages when downstream operations fail.
Location	- `.../src/index.tsx:380`
Description	The ERC20 token import handler within `onUserInput` checks the submitted address with a simple length comparison, which only verifies that the string is 42 characters long but does not validate hex format, the 0x prefix, or EIP-55 checksum. For example, a string like `0000000000000000000000000000000000000000ab` (42 characters, no 0x prefix) or `0xZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ` (42 characters, invalid hex) would pass this validation. The subsequent `getERC20Details(string)` call will fail when trying to query the address on-chain, but the error is generic rather than a clear validation message.
Mitigation	Use `ethers.isAddress()` for proper Ethereum address validation, which checks format, length, and optional EIP-55 checksum in a single call.

Unvalidated `JSON.parse` on Untrusted Input

ID	SAY-07
Status	Fixed
Risk	Low
Business Impact	A Malformed JSON will throw a `SyntaxError` that propagates as an unhandled RPC error, potentially leaking stack trace information to the calling dapp.
Location	- `.../src/index.tsx:546`
Description	In the decrypt RPC handler, `JSON.parse(encryptedValue)` is called directly on the value supplied by the calling dapp to extract ciphertext and r-fields. There is no prior validation that the parameter is a valid JSON or that the parsed result contains the expected fields before destructuring. While the broader RPC switch statement may catch this error, the resulting error message will be a raw `SyntaxError` from `JSON.parse` like "Unexpected token X in JSON at position Y", rather than a clean, context-appropriate error. This can leak internal implementation details to the calling dapp and provides a poor developer experience for legitimate integrators.
Mitigation	Wrap the `JSON.parse` call in a try-catch block and return a clean, descriptive error message.



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