SAYFER

Metamask Snap Audit Report for LiquidLink

Testers:

1. Or Duan
2. Avigdor Sason Cohen

Table of Contents

Management Summary

LiquidLink contacted Sayfer Security in order to perform penetration testing on their Iota Metamask snap in July 2025.

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

Over the research period of 2 weeks, we discovered 5 vulnerabilities in the system.

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

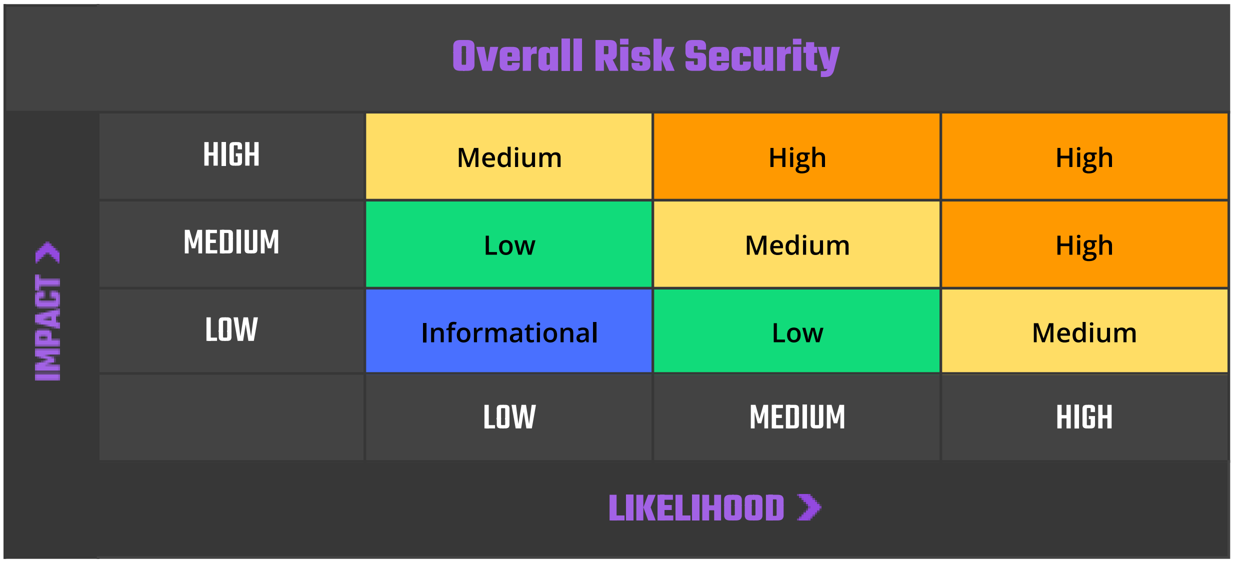
Risk Methodology

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:

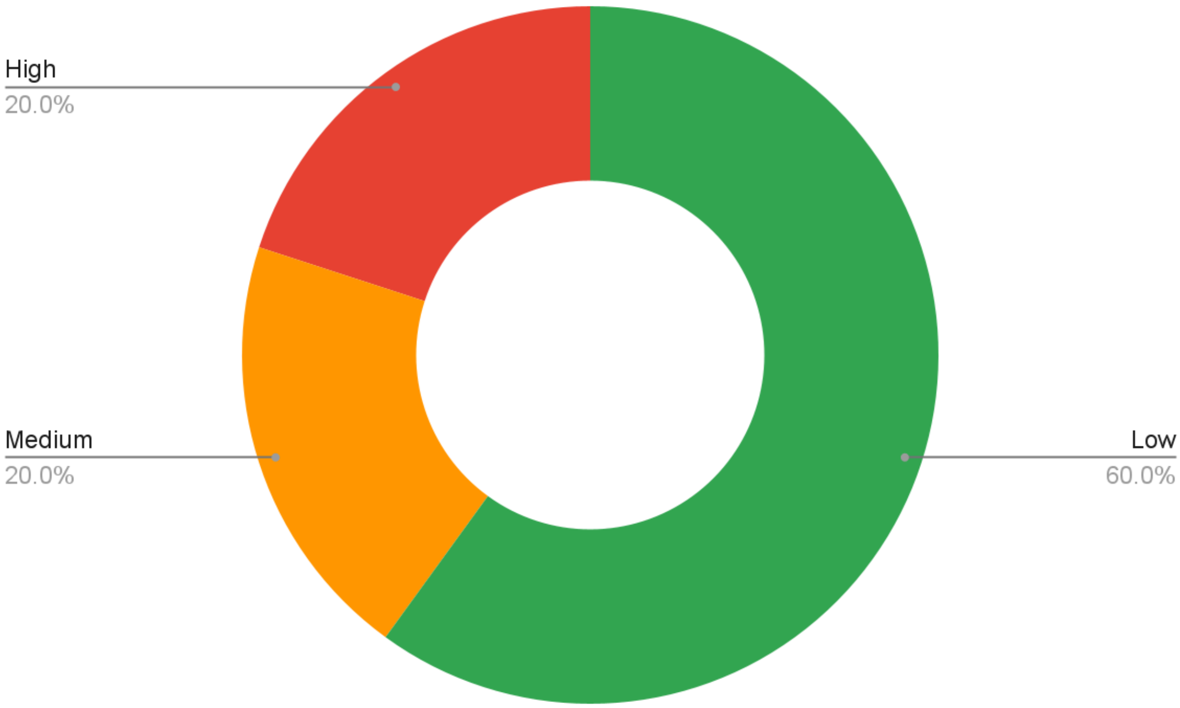


Overall Risk Security

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	3	1	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.

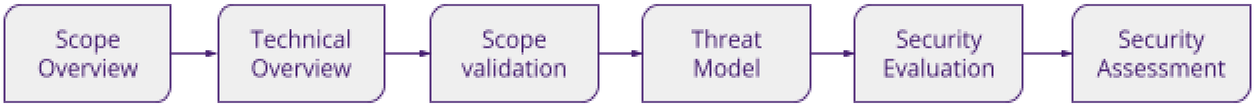
Approach

Introduction

LiquidLink 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 LiquidLink'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:

- LiquidLink’s MetaMask Snap
    - **Audit commit:** `e92829c642967285de8531d15d2a6be13551d19b`
    - **Fixes commit:**

Our tests were performed from 27/07/2025 to 10/08/2025.

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 that the largest current threat to the system is the potential for attackers to siphon funds from the user’s wallet.

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

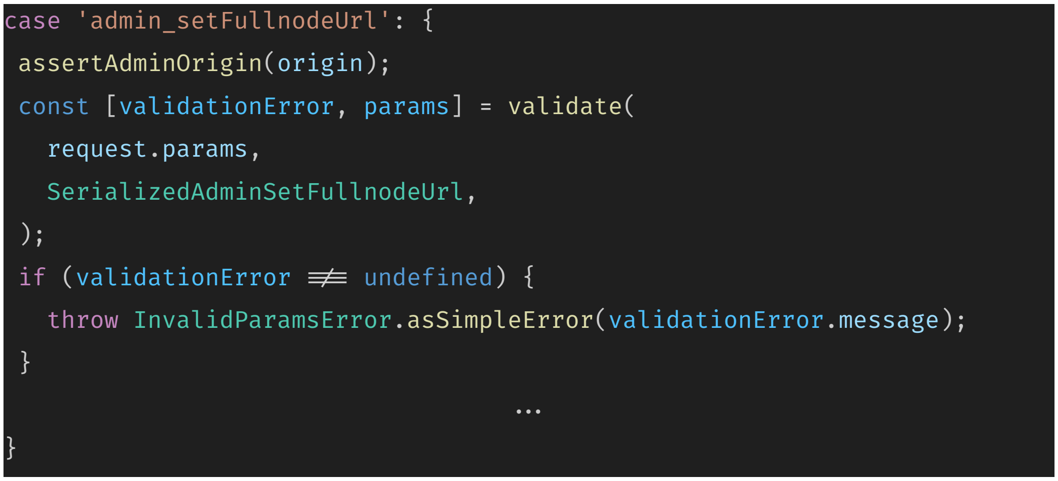
Client Side Testing	Test Name	Status
WSTG-CLNT-01	Testing for DOM-Based Cross Site Scripting	Pass
WSTG-CLNT-02	Testing for JavaScript Execution	Pass
WSTG-CLNT-03	Testing for HTML Injection	Pass
WSTG-CLNT-04	Testing for Client Side URL Redirect	Pass
WSTG-CLNT-05	Testing for CSS Injection	Pass
WSTG-CLNT-06	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

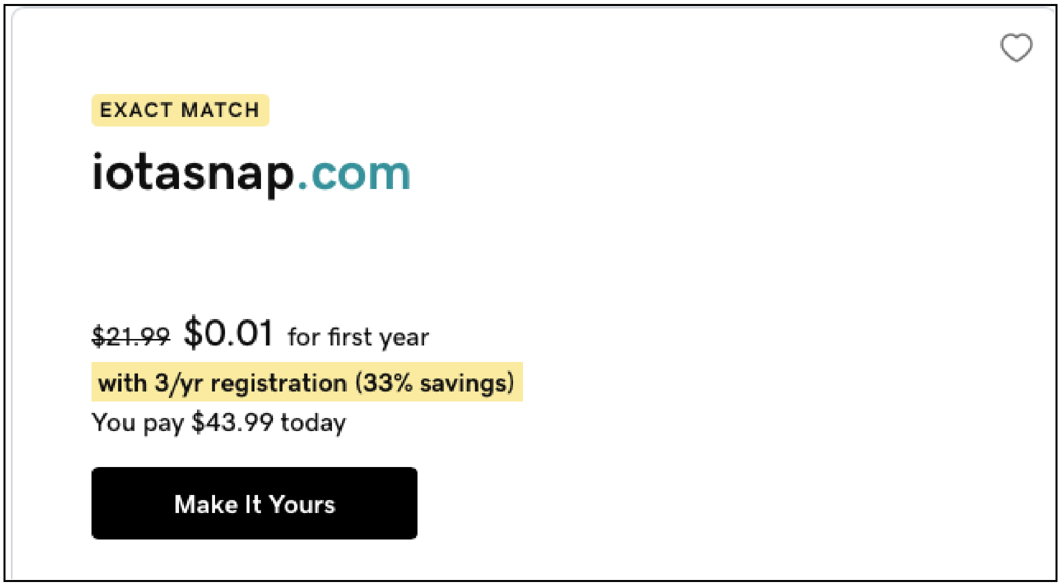
Arbitrary Fullnode URL Injection

ID	SAY-01
Status	Open
Risk	High
Business Impact	If an attacker acquires the whitelisted domain, they can bypass the admin boundary to set arbitrary fullnode URLs, corrupting the snap's view of the chain and tricking users into approving malicious or misrepresented operations.
Location	- `index.tsx; admin_setFullnodeUrl`
Description	By design, the snap gates admin functionality like setting the fullnode URL through the `admin_setFullnodeUrl` method, solely by checking the origin against hardcoded values, including `https://iotasnap.com`. If an attacker is able to acquire that domain, they can host a malicious page that will pass `assertAdminOrigin` and invoke admin-only methods like `admin_setFullnodeUrl`. ● `index.tsx:271-279`



```
case 'admin_setFullnodeUrl': {
 assertAdminOrigin(origin);
 const [validationError, params] = validate(
   request.params,
   SerializedAdminSetFullnodeUrl,
 );
 if (validationError !== undefined) {
   throw InvalidParamsError.asSimpleError(validationError.message);
 }
                                    ...
}
```

Because the fullnode URL is accepted without validation and user interaction, the attacker can then point the snap at arbitrary endpoints and manipulate all downstream behavior. Indeed, we found the whitelisted domain is not registered, and can be acquired by potential attackers.

	 EXACT MATCH iotasnap.com ~~$21.99~~ $0.01 for first year with 3/yr registration (33% savings) You pay $43.99 today
Mitigation	We recommend ensuring that any admin origin domain is actually owned and provably controlled. Additionally even if the domain is trusted, it is suggested to inform the user with some dialog box, that his RPC URL was changed.

Lack of Input Validation for RPC Parameters

ID	SAY-02
Status	Open
Risk	Medium
Business Impact	Malformed RPC payloads can propagate arbitrary data into downstream logic like transaction construction, UI rendering or error handling, making it easier to craft payloads that appear benign in the UI while behaving differently internally.
Location	- `types.ts; validate<TData>(unknown, unknown => TData)`

Description

The validation in most of the state changing methods is a no-op cast.

- `types.ts:129-132`

```
export const SerializedIotaSignPersonalMessageInput = (
 params: unknown,
): SerializedIotaSignPersonalMessageInput =>
 params as SerializedIotaSignPersonalMessageInput;
```

And `validate<TData>( ... )` itself just calls the schema and checks structural correctness. There is no type enforcement, and no sanitization.

- `types.ts:110-125`

```
export function validate<TData>(
 params: unknown,
 schema: (params: unknown) => TData,
): [Error | undefined, TData] {
 try {
   // This is a simplified version since we don't know the exact
validation library
   // you might be using. If you're using zod, you would use
schema.parse(params)
   const result = schema(params);
   return [undefined, result];
 } catch (error) {
   return [
     error instanceof Error ? error : new Error('Validation failed'),
     {} as TData,
   ];
 }
}
```

Mitigation	Replace the current schema stubs with a real parser that strictly enforces required fields, types, allowed values, and rejects unexpected fields Additionally make sure that any subsequent usage assumes a properly validated shape, eliminating reliance on *any* where possible.

Transaction Mutability between Dry Run and Execution

<table>
<tr><td>ID</td><td>SAY-03</td></tr>
<tr><td>Status</td><td>Open</td></tr>
<tr><td>Risk</td><td>Low</td></tr>
<tr><td>Business Impact</td><td>Users can approve a dry-run transaction but a different transaction may be executed, leading to unintended state changes or asset movement.</td></tr>
<tr><td>Location</td><td>

- `index.tsx; signAndExecuteTransaction`

</td></tr>
<tr><td>Description</td><td>

Dry runs are performed on the result of `buildTransactionBlock(string, Transaction, string)`, which may mutate the transaction block (e.g. setting the sender) and produces canonical bytes (`transactionBlockBytes`).

- `util.ts:142-144`

```
if (!transactionBlock.getData().sender) {
 transactionBlock.setSender(sender);
}
```

However, after user confirmation, execution calls `signAndExecuteTransaction(IotaClient, any, any, any)` with the original `input.transactionBlock` instead of the built one. There is no binding or fingerprint comparison between what was dry-run and what is actually executed, so the executed transaction can diverge.

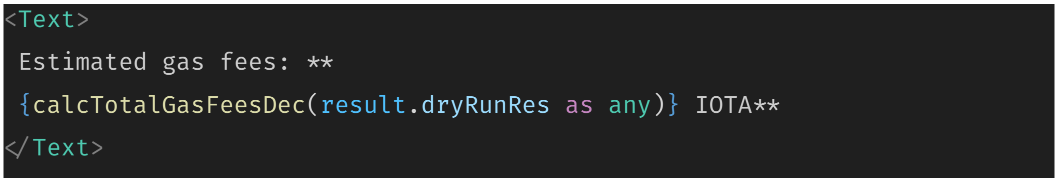
- `index.tsx:255-263`

```
const res = await signAndExecuteTransaction(
 client,
 input.transactionBlock as any,
 input.requestType,
 input.options,
);
const ret = res as any as IotaSignAndExecuteTransactionOutput;
return ret;
}
```

</td></tr>
<tr><td>Mitigation</td><td>We recommend exactly executing the transaction that was dry-run: pass the built/mutated transaction block or its serialized bytes to <code>signAndExecuteTransaction(…)</code>, or hash the dry-run bytes and compare against the execution serialization, aborting if they differ.</td></tr>
</table>

Misleading Estimated Gas Fees

ID	SAY-04
Status	Open
Risk	Low
Business Impact	If the gas estimation is missing, the UI shows “0 IOTA” as the fee, misleading the user into approving a transaction under the false assumption that it is free.
Location	- `util.ts; calcTotalGasFeesDec(DryRunTransactionBlockResponse)`

Description	`calcTotalGasFeesDec( ... )` returns the literal string '0' when `dryRunRes ?. effects ?. gasUsed` is *`undefined`*, returning “no cost” rather than “estimate unavailable”. That value is then rendered in the confirmation dialog with no differentiation or warning when the underlying data was missing. The absence of `gasUsed` silently collapses into a zero-fee presentation.
	- `index.tsx:241-244`
	
	`<Text>`
	` Estimated gas fees: **`
	` {calcTotalGasFeesDec(result.dryRunRes as any)} IOTA**`
	`</Text>`

Mitigation	We recommend distinguishing between “zero fee” and “fee estimate unavailable”.

Inadequate Control Character Detection

<table>
<tr><td>ID</td><td>SAY-05</td></tr>
<tr><td>Status</td><td>Open</td></tr>
<tr><td>Risk</td><td>Low</td></tr>
<tr><td>Business Impact</td><td>Messages containing invisible or formatting characters can be shown as “human-readable” while hiding malicious intent, causing users to approve and sign payloads they misunderstand.</td></tr>
<tr><td>Location</td><td>- `index.tsx; signPersonalMessage`</td></tr>
<tr><td>Description</td><td>The code uses the regex `/[�-   -]/u` to detect control characters and falls back to base 64 encoding when they appear. That pattern only covers a subset of C0 control characters and omits others (and does not address Unicode format characters like bidi overrides or zero-width characters) that can be abused for visual deception.

- `index.tsx:52-55`

<pre>
if (/[�- -]/u.test(decodedMessage)) {
 decodedMessage = Buffer.from(input.message).toString('base64');
 info = `**${origin}** is requesting to sign the following message
(base64 encoded):`;
}
</pre>

Malicious actors could inject Unicode control characters like right-to-left override or zero-width spaces to manipulate how messages appear to users. Additionally, Certain control characters could break the MetaMask UI dialog rendering.</td></tr>
<tr><td>Mitigation</td><td>Replace the heuristic with robust detection, testing for all Unicode control and format characters using property escapes, like `/\p{Cc}|\p{Cf}/u` and explicitly handling known spoofing vectors like bidi overrides and zero-width characters. Alternatively, if such characters will be found in the signed string - you could include the extra string to the message informing users that something seems to be wrong or malicious.</td></tr>
</table>

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