SaucerSwap ERC20Wrapper Security Review

Auditors
Kaden, Security Researcher

August 25, 2025

1 Executive Summary

Over the course of 1 day in total, SaucerSwap engaged with Kaden to review saucerswap-erc20-wrapper.

Metadata

Repository	**Commit**
saucerswap-erc20-wrapper	739cecc

Summary

Type of Project	Wrapper
Timeline	August 22nd, 2025
Methods	Manual Review

Total Issues

Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	1
Gas Optimizations	1

Contents

2 Introduction

SaucerSwap is an open source and non-custodial crypto trading protocol on the Hedera network.

Disclaimer: This review does not make any warranties or guarantees regarding the discovery of all vulnerabilities or issues within the audited smart contracts. The auditor shall not be liable for any damages, claims, or losses incurred from the use of the audited smart contracts.

3 Findings

3.1 Low Risk

3.1.1 Smart contract rent is shared between all wrapped tokens

Severity: Low

Context: `ERC20Wrapper.sol#L17`

Description:

`ERC20Wrapper.sol` operates as a singleton contract, holding state for all wrapped tokens. While this architecture is gas efficient, it may complicate the responsibility of paying smart contract rent. As defined in the Hedera documentation, "Smart contract rent is a recurring payment mechanism designed to maintain resource allocation and is required for contracts to remain active on the network." In case rent goes unpaid, contract state will be lost, effectively burning tokens and breaking surrounding logic.

Since `ERC20Wrapper.sol` is permissionless, anyone can create tokens, even if they are not being used. This opens up the possibility of a griefing vector wherein an attacker creates spam tokens to increase rent costs without paying for their share of the costs. Furthermore, this kind of state bloat may occur regardless of intent, e.g., with many low value tokens being created and going unused over time, yet increasing recurring rent costs.

Recommendation:

Consider moving to a factory architecture whereby each ERC20 wrapper is a separate contract. Alternatively, consider making `ERC20Wrapper.create` a permissioned function to limit state bloat.

SaucerSwap: Acknowledged.

3.2 Gas Optimizations

3.2.1 Redundant decimal bound validation

Severity: Gas Optimization

Context: `HTSLib.sol#L36`

Description:

In ERC20Wrapper.create, we clamp the htsDecimals to a maximum of MAX_DECIMALS (8) before passing this value to HTSLib.create:

```
htsDecimals = erc20Decimals > MAX_DECIMALS ? MAX_DECIMALS : erc20Decimals;

htsToken = HTSLib.create(name, symbol, htsDecimals);
```

In HTSLib.create, we validate that the provided htsDecimals is less than or equal to type(int32).max:

```
require(htsDecimals <= uint256(int256(type(int32).max)), "DECIMALS_OVERFLOW");
```

Since we've already clamped htsDecimals, we know that it can't exceed type(int32).max, therefore the above require statement is redundant.

Recommendation:

Consider removing the redundant require statement in HTSLib.create:

```
-require(htsDecimals <= uint256(int256(type(int32).max)), "DECIMALS_OVERFLOW");
```

SaucerSwap: Acknowledged.