HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: SaucerSwap
Date: July 12th, 2022

This document may contain confidential information about IT systems and the intellectual property of the Customer as well as information about potential vulnerabilities and methods of their exploitation.

The report containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities are fixed — upon a decision of the Customer.

Document

Name	Smart Contract Code Review and Security Analysis Report for SaucerSwap
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type	HTS token; Vesting
Platform	Hedera
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	saucerswap.finance
Timeline	30.05.2022 - 12.07.2022
Changelog	21.06.2022 - Initial Review 12.07.2022 - Second Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by SaucerSwap (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project is smart contracts in the repository:

Initial review scope

Repository:
    https://github.com/littletarzan/saucerswap-vesting
Commit:
    985e318061e9b89ea19eca44919297d6b3a2d4b3
Technical Documentation:
    Type: Whitepaper
    Link
    Link

Integration and Unit Tests: Yes
Contracts:

```
File: ./contracts/hip-206/HederaResponseCodes.sol
SHA3: b97c19959a7cc7d63470cf66f7768b445624e50e8b38b638fb984537a98d74b8

File: ./contracts/hip-206/HederaTokenService.sol
SHA3: 0d4a77bbff112517715a1d467e717b22dbc49532cc776c4f70bd1d02d1382a71

File: ./contracts/hip-206/IERC20.sol
SHA3: 33406db4e10278eec17fbfc497472d9923e2f9b1ed978449af7a0ca20e5092ea

File: ./contracts/hip-206/IHederaTokenService.sol
SHA3: 87f10b69dc9dad41ff3a7d4caff51fbf47c2bfaa0b09974d433b9bbff21fa97b

File: ./contracts/hip-206/SafeHederaTokenService.sol
SHA3: 621bee8615085773d504cd3e60fac8da868b0bd196b55061dc8a88128c350795

File: ./contracts/Migrations.sol
SHA3: f38ad4185f0fa410f3427a0bae9195f29bf1c8806f1a019cc727d7c39b53811d

File: ./contracts/OpenZeppelin/Ownable.sol
SHA3: 0d5d96b5a497a0974e267c6a6e0e4e982fe162dcea907c93fac7502801ab4dc0

File: ./contracts/OpenZeppelin/SafeCast.sol
SHA3: 9a15f06755e718c19fe4aff553d057c4f7e5d2fdf3a9e7cdc077f32caccb6eec

File: ./contracts/PaymentSplitter.sol
SHA3: 84b930ab8d11f227395fa636723a6ae0d7f600ca7f342e5d96f8109b844910c9

File: ./contracts/VestingWallet.sol
SHA3: ddb47b0870e086a3ed4ba70245dac9d379a9fd249777205b10edbfa6afd070f4
```

Second review scope

Repository:
    https://github.com/littletarzan/saucerswap-vesting/tree/remediation-hacken
Commit:

bfbd41f7ddc9b0c4808ba0f994a4562885fee4db

Technical Documentation:

Type: Whitepaper
Link
Link

Integration and Unit Tests: No
Deployed Contracts Addresses: No
Contracts:

File: ./contracts/hip-206/HederaResponseCodes.sol
SHA3: 7448eb04b088a187e632eda32d0843216ba46bf7ab04ab574a02a0cb4f034ff3

File: ./contracts/hip-206/HederaTokenService.sol
SHA3: 5dcce08148d10d07df71efccded44ec8fbb0fb0d00c8a17b02d5a340b6a32450

File: ./contracts/hip-206/IERC20.sol
SHA3: 92116e0dc5c92561e18cd0e72573e7e90e39ba00666f5fd957694df9f16ef349

File: ./contracts/hip-206/IHederaTokenService.sol
SHA3: f7bef2808a894e44a6e4b6283c9b99bc55a56e66be8291e419d250a4dcb129ba

File: ./contracts/hip-206/SafeHederaTokenService.sol
SHA3: d212fde46d5824da0091e9cc77719f2bb8cbcc80535162d35f0d17d6c68c9d33

File: ./contracts/Migrations.sol
SHA3: 2087b82b45792dd7a63f3caaec0f3fe32b71b3e9a26352147ccc0b43476045de

File: ./contracts/OpenZeppelin/Ownable.sol
SHA3: c8d86d0d0ddd6dae5419b32ff5b399ff43e6c56793e23109500974ca6e11a3b5

File: ./contracts/OpenZeppelin/SafeCast.sol
SHA3: 0eff98b0564702e56534d27ae9f4d1d860157aac5a4e48efe490e21317ea9e8b

File: ./contracts/PaymentSplitter.sol
SHA3: 0f44743af6a8495f6658e30b1b7a3fda2bccbbc69d4a95777f34221b5cf905d3

File: ./contracts/VestingWallet.sol
SHA3: 454a312aedd5739bd31052e8694161887e1f1d40e741769d3b6a3253aac77780

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions.
Medium	Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution.

Executive Summary

The score measurement details can be found in the corresponding section of the methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**. A comprehensive whitepaper is provided.

Code quality

The total CodeQuality score is **7** out of **10**. Unit tests were provided, but they were not running.

Architecture quality

The architecture quality score is **10** out of **10**.

Security score

As a result of the audit, the code contains **1** low severity issue. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.7**.

1
2
3
4
5
6
7
8
9
10

The final score

Checked Items

We have audited provided smart contracts for commonly known and more specific vulnerabilities. Here are some of the items that are considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Not Relevant
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Passed
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Uninitialized Storage Pointer	SWC-109	Storage type should be set explicitly if the compiler version is < 0.5.0.	Not Relevant
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Not Relevant
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Passed
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless it is required.	Passed
Race	SWC-114	Race Conditions and Transactions Order	Passed

Conditions		Dependency should not be possible.	
Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Passed
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Passed
Signature Unique Id	SWC-117 SWC-121 SWC-122	Signed messages should always have a unique id. A transaction hash should not be used as a unique id.	Not Relevant
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of unused variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP standards violation	EIP	EIP standards should not be violated.	Not Relevant
Assets integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions.	Passed
User Balances manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant
Token Supply manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the customer.	Not Relevant
Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There	Passed

		should not be any cases when execution fails due to the block Gas limit.	
Style guide violation	**Custom**	Style guides and best practices should be followed.	Passed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Failed
Stable Imports	**Custom**	The code should not reference draft contracts, that may be changed in the future.	Passed

System Overview

Saucerswap Vesting is a vesting system with the following contracts:

- *PaymentSplitter* – a contract that allows to split token payments among a group of accounts, designed for Hedera SAUCE tokens.
  It has the following attributes:
  - payees array: the addresses array of the payees
  - shares array: keeps the share amount for each payee
  - _tokenAddr: payment token address
- *VestingWallet* – a contract that controls the vesting Eth and ERC20 tokens for a beneficiary address.
- Migrations – a contract that sets the last completed migration.
- HederaTokenService – an abstract contract that allows associating a provided address to a provided Hedera token and allows transferring tokens.
- SafeHederaTokenService – an abstract contract that allows associating a provided address to a provided Hedera token and transferring tokens safely with checking the result.
- HederaResponseCodes – an abstract contract that declares all the Hedera Token Service's response codes.

Privileged roles

- The owner of the PaymentSplitter contract can add a payee or change a payee's share amount in the contract.
- The owner of the VestingWallet contract can change the beneficiary address.

Findings

🟥🟥🟥🟥 Critical

1. Total shares are not getting updated

In _addPayee function, the owner is able to change a payee’s share amount, but the total share is not getting updated.

When the payee’s shares are adjusted, the unchanged _totalShares value will cause wrong calculations for the release. This may lead some users to lose their funds.

File: ./contracts/PaymentSplitter.sol

Contract: PaymentSplitter

Function: adjustSharesPayee

Recommendation: Update _totalShares after a payee’s share is changed.

Status: Fixed (Revised commit : bfbd41f7ddc9b0c4808ba0f994a4562885fee4db)

🟨🟨🟨 High

No high severity issues were found.

🟩🟩 Medium

No medium severity issues were found.

🟦 Low

1. Functions declared as public

Although some functions are not called internally, their visibility is declared as public.

Public visibility requires more Gas than the external.

File: ./contracts/PaymentSplitter.sol, ./contracts/VestingWallet.sol

Contract: PaymentSplitter, VestingWallet

Functions: PaymentSplitter.tokenAddr, PaymentSplitter.totalShares, PaymentSplitter.shares, PaymentSplitter.payee, PaymentSplitter.release, PaymentSplitter.getBal, VestingWallet.tokenAddr, VestingWallet.release

Recommendation: Replace public visibilities with external.

Status: Fixed (Revised commit : bfbd41f7ddc9b0c4808ba0f994a4562885fee4db)

2. Unlocked compiler version

Some contracts in the project are not locked to a stable compiler version.

It leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can cause confusion when debugging because compiler-specific defects can appear in the codebase and be difficult to spot across numerous compiler versions rather than a single one.

File: ./contracts/PaymentSplitter.sol, ./contracts/hip-206/HederaResponseCodes.sol, ./contracts/hip-206/HederaTokenService.sol, ./contracts/hip-206/IHederaTokenService.sol, ./contracts/hip-206/SafeHederaTokenService.sol, ./contracts/Migrations.sol

Contract: PaymentSplitter, HederaResponseCodes, HederaTokenService, IHederaTokenService, SafeHederaTokenService, Migrations

Functions: -

Recommendation: Lock the compiler version to a recent one and use the same compiler version for all contracts.

Status: Fixed (Revised commit : bfbd41f7ddc9b0c4808ba0f994a4562885fee4db)

3. Using experimental ABI encoder

ABI encoder v2 is not considered experimental anymore. It can be selected via pragma abicoder v2 since Solidity 0.7.4 and it is already activated by default starting from Solidity 0.8.0.

No need to define it explicitly. Extra-long code consumes more Gas and can decrease the code readability.

File: ./contracts/hip-206/HederaTokenService.sol, ./contracts/hip-206/IHederaTokenService.sol, ./contracts/hip-206/SafeHederaTokenService.sol

Contract: HederaTokenService, IHederaTokenService, SafeHederaTokenService

Functions: -

Recommendation: Remove the experimental ABI encoder.

Status: Fixed (Revised commit : bfbd41f7ddc9b0c4808ba0f994a4562885fee4db)

4. Outdated solidity compiler version

Using an outdated compiler version can be problematic, especially if publicly disclosed bugs and issues affect the current compiler version.

File: ./contracts/hip-206/IERC20.sol, ./contracts/PaymentSplitter.sol, ./contracts/VestingWallet.sol

Contract: IERC20, PaymentSplitter, VestingWallet

Functions: -

Recommendation: Change the compiler version with the recent one.

Status: Fixed (Revised commit : bfbd41f7ddc9b0c4808ba0f994a4562885fee4db)

5. Unused variables

Looks like only SUCCESS, UNKNOWN used. Other constants were unused, making the code overwhelmed and less readable.

Contract: HederaResponseCodes.sol

Recommendation: Review and clean up the code.

Status: Fixed (Revised commit : bfbd41f7ddc9b0c4808ba0f994a4562885fee4db)

6. Datatype overwhelmed

Datatype for *_start* and *_duration* variables are *uint64* while functions, which return them have *uint256* type. This makes the code a little bit confused.

Contract: VestingWallet.sol

Function: start, duration

Recommendation: Use the same datatype.

Status: Fixed (Revised commit : bfbd41f7ddc9b0c4808ba0f994a4562885fee4db)

7. Style guide violation

Contracts do not follow the Solidity code style guide.

Contracts: HederaTokenService.sol, Ownable.sol, PaymentSplitter.sol, VestingWallet.sol

Recommendation: Follow the official Solidity code style guide.

Status: Reported (Revised commit : bfbd41f7ddc9b0c4808ba0f994a4562885fee4db)

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The audit makes no statements or warranties on the security of the code. It also cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the audit cannot guarantee the explicit security of the audited smart contracts.