HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: SaucerSwap
Date: July 14th, 2022

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for SaucerSwap
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type	Staking
Platform	Hedera Network
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	saucerswap.finance
Timeline	20.06.2022 - 14.07.2022
Changelog	23.06.2022 - Initial Review 12.07.2022 - Second Review 14.07.2022 - Third Review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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/vea-vecturne/saucerswap-farm
Commit:
8271c1189
Technical Documentation:
Type: Technical Requirements
Link

Integration and Unit Tests: No
Deployed Contracts Addresses: No

Contracts:
File: ./contracts/hedera/HederaResponseCodes.sol
SHA3: b97c19959a7cc7d63470cf66f7768b445624e50e8b38b638fb984537a98d74b8

File: ./contracts/hedera/HederaTokenService.sol
SHA3: 181617dade3149ef0a055773b4931f783ca04bf97ccabeabdb04d7fc3471b671

File: ./contracts/hedera/IExchangeRate.sol
SHA3: 653b75ab25c7cc7103a8f09f14bfa8b6e0b1dccce713757757636a38ae295798

File: ./contracts/hedera/IHederaTokenService.sol
SHA3: 3d41eb8cba1880a1cddd3d6303c7f03af0e03e35cab7fe9da12bb51bbf87575c

File: ./contracts/hedera/SafeHederaTokenService.sol
SHA3: ba993bddc2f306b9d7e76c2e3c9734a37acb379742b7c911c4936f87c63c10f1

File: ./contracts/MasterChef.sol
SHA3: bb75db3b0d2169120c16f76eea27414e16b3dd0c693e05521b2109e515307f80

File: ./contracts/Migrations.sol
SHA3: dc1cdf247bdfe7c6d67b49b03610547befd29b0886df43f70f5b12274da76a84

File: ./contracts/OpenZeppelin/IERC20.sol
SHA3: ccfd26f02b50ff9b74f99bfabc14181b711f3add33fc50868d8815baa0f4ccde

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

File: ./contracts/OpenZeppelin/ReentrancyGuard.sol
SHA3: 91d926de6af7b89e7e9a90ffc979783e397108ee0f2092470288e922bd81c742

File: ./contracts/OpenZeppelin/SafeCast.sol
SHA3: 30ac408e20b2c7c90405d41b30ba09738e8e76bb6d1b8ac38a8c79ad535d97d8

File: ./contracts/OpenZeppelin/SafeMath.sol
SHA3: d5f8810f0333d156c86735b81107f745fc89191a39f29fa066e6cc8f869a8082

Second review scope

Repository:
https://github.com/vea-vecturne/saucerswap-farm
Commit:
78531f3a4de8d9de13bc7c614f67e62299f7e4bb
Technical Documentation:
Type: Technical Requirements
Link

Type: Functional Requirements
Link
Integration and Unit Tests: No
Deployed Contracts Addresses: No
Contracts:
File: ./contracts/hedera/HederaResponseCodes.sol
SHA3: 306b59b0b3eaac51db94c8d488084f6b4843ef5703cc5c63c43c7cb9b7688004

File: ./contracts/hedera/HederaTokenService.sol
SHA3: dfd31c6f1cd6c373b1a04fc5d83609ff1a540b911436369dc9454c633f548c3f

File: ./contracts/hedera/IExchangeRate.sol
SHA3: ae8804ad840ae9bd601065910c3f9112a33540d41b4de8a5a6611219536cf7fa

File: ./contracts/hedera/IHederaTokenService.sol
SHA3: 37a1210e57fb0389fda6d522ad9465251e4a92e9f0b905d58e92554dfc3ba4f0

File: ./contracts/hedera/SafeHederaTokenService.sol
SHA3: 541551e3b1ceba86e6851351f2d28abce5ec88d01a113b91ecd4cc82831a03a8

File: ./contracts/MasterChef.sol
SHA3: f49b7d49e80bce38eb402997b02c0ba0268102b36987c44228dcb3a426a9428d

File: ./contracts/Migrations.sol
SHA3: dc1cdf247bdfe7c6d67b49b03610547befd29b0886df43f70f5b12274da76a84

File: ./contracts/OpenZeppelin/IERC20.sol
SHA3: 114ebe2ab981a2a7eb960c3edc54e714b79df80eebf6ae99ba85a05b6a15d149

File: ./contracts/OpenZeppelin/Ownable.sol
SHA3: e6114abd5265b17b0c4049ebe6a3460fbdc5eca90ffdd5cf77b0389f244fc038

File: ./contracts/OpenZeppelin/ReentrancyGuard.sol
SHA3: 284f545522fd3fb3cf32dfdb7f5f3b6b9346973c89df38fb0c26d565b997da26

File: ./contracts/OpenZeppelin/SafeCast.sol
SHA3: 6106955f85e2856e52053be8ea1f216f4ede7fe62566bbd068cdac184cb96e14

Third review scope

Repository:
https://github.com/vea-vecturne/saucerswap-farm
Commit:
d5e194537c4508574fa608140859e339a198a43c
Technical Documentation:
Type: Technical Requirements
Link

Type: Functional Requirements
Link
Integration and Unit Tests: Yes
Deployed Contracts Addresses: No
Contracts:

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

```
File: ./contracts/hedera/HederaResponseCodes.sol
SHA3: 306b59b0b3eaac51db94c8d488084f6b4843ef5703cc5c63c43c7cb9b7688004

File: ./contracts/hedera/HederaTokenService.sol
SHA3: dfd31c6f1cd6c373b1a04fc5d83609ff1a540b911436369dc9454c633f548c3f

File: ./contracts/hedera/IExchangeRate.sol
SHA3: ae8804ad840ae9bd601065910c3f9112a33540d41b4de8a5a6611219536cf7fa

File: ./contracts/hedera/IHederaTokenService.sol
SHA3: 37a1210e57fb0389fda6d522ad9465251e4a92e9f0b905d58e92554dfc3ba4f0

File: ./contracts/hedera/SafeHederaTokenService.sol
SHA3: 541551e3b1ceba86e6851351f2d28abce5ec88d01a113b91ecd4cc82831a03a8

File: ./contracts/MasterChef.sol
SHA3: 3fd229b795be71923cb787ae7a4640aa60e7b1f14cec9b00a60d8ac51705aa5c

File: ./contracts/Migrations.sol
SHA3: dc1cdf247bdfe7c6d67b49b03610547befd29b0886df43f70f5b12274da76a84

File: ./contracts/OpenZeppelin/IERC20.sol
SHA3: 114ebe2ab981a2a7eb960c3edc54e714b79df80eebf6ae99ba85a05b6a15d149

File: ./contracts/OpenZeppelin/Ownable.sol
SHA3: e6114abd5265b17b0c4049ebe6a3460fbdc5eca90ffdd5cf77b0389f244fc038

File: ./contracts/OpenZeppelin/ReentrancyGuard.sol
SHA3: 284f545522fd3fb3cf32dfdb7f5f3b6b9346973c89df38fb0c26d565b997da26

File: ./contracts/OpenZeppelin/SafeCast.sol
SHA3: 6106955f85e2856e52053be8ea1f216f4ede7fe62566bbd068cdac184cb96e14
```

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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**.

Code quality

The total CodeQuality score is **9** out of **10**. Code is not fully following the official style guide.

Architecture quality

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

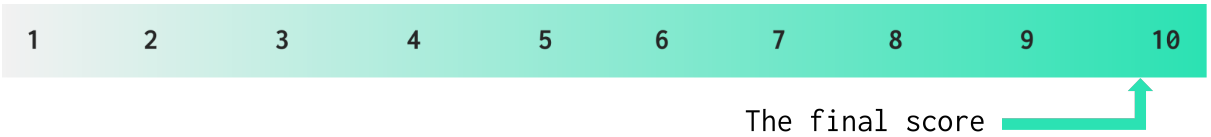
Security score

As a result of the audit, the code contains **2** low severity issues. 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: **9.9**.



Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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.	Passed
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.	Passed
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.	Not Relevant
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	Not Relevant

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.	Passed
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.	Passed
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.	Passed
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.	Failed
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.	Passed
Stable Imports	**Custom**	The code should not reference draft contracts, that may be changed in the future.	Passed

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

System Overview

Farm Repo is a staking system with the following contracts:

- *MasterChef* — staking contract, which allows users to stake tokens in the specific pool and earn rewards.
  It has the following attributes:
  - Dev address - developer address to pay a share.
  - Rent payer address - address for paying commissions upon deposit.
  - Sauce per second - how much SAUCE will be generated per second.
  - Hbar per second - how much HBAR will be generated per second.
  - Maximum SAUCE supply.
  - Deposit fee.
- *Migrations.sol* - Truffle default contract to keep track migrations.
- *IERC20.sol* - standard ERC20 interface.
- *Ownable* - endow with facilities to validate ownership permissions, transfer ownership.
- *ReentrancyGuard* - protection from reentrance attacks.
- *HederaResponseCodes* - contain all response codes from Hedera.
- *HederaTokenService* - used as a bridge between Hedera and smart contracts, allow to call Hedera methods from the pre-compiled contract as a mint, association and transferring tokens, getting exchange rates.
- *SafeHederaTokenService* - decorator for *HederaTokenService* which checks transaction results and will revert them in case they are unsuccessful.

Privileged roles

- The owner can:
  - set SAUCE token address.
  - set deposit fee address.
  - set how many HBAR per second will be generated.
  - set how many SAUCE per second will be generated.
  - set maximum SAUCE supply.
  - add pools.
  - change pool allocation.
  - set developer address.
  - set rent payer address to collect commissions from deposits.
  - transfer, renounce ownership.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Findings

Critical

No critical severity issues were found.

High

1. **Potential DoS**

   If a ‘rentPayer’ address is a contract address and this contract has a fallback or receive function, the transfer will fail due to the 2300 Gas limit.

   Possible DoS.

   Contract: MasterChef.sol

   Function: deposit

   Recommendation: Ensure the rent payer address is externally owned or use a call method instead of send.

   Status: Fixed (Revised commit: 78531f3a4de8d9de13bc7c614f67e62299f7e4bb)

2. **Potential DoS**

   Specified functions iterate over all pools. Moreover, *massUpdatePools* changes state every iteration.

   Possible DoS if the number of pools is large enough.

   Contract: MasterChef.sol

   Functions: massUpdatePools, checkForDuplicate

   Recommendation: Do not iterate over all pools or limit the number of total pools.

   Status: Fixed (Revised commit: d5e194537c4508574fa608140859e339a198a43c)

Medium

1. **Floating pragma**

   The contracts use quite a wide floating pragma range >=0.4.9 < 0.9.0.

   Moreover, Solidity 0.9.0 is not yet released.

   Contract: all

   Recommendation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment.

   Status: Fixed (Revised commit: 78531f3a4de8d9de13bc7c614f67e62299f7e4bb)

2. **Using SafeMath**

SafeMath is generally not needed starting with Solidity 0.8.

Contract: MasterChef.sol

Recommendation: Remove SafeMath.

Status: Fixed (Revised commit: 78531f3a4de8d9de13bc7c614f67e62299f7e4bb)

3. **Unchecked return value of a call method**

   The result of the call to contract is not checked on lines 341 and 343.

   Therefore users can lose their rewards in case of a failed transaction.

   Contract: MasterChef.sol

   Function: safeHBARTransfer

   Recommendation: Check the return value of a call method and revert transaction in case it is *false*.

   Status: Mitigated (with customer notice) (Revised commit: 78531f3a4de8d9de13bc7c614f67e62299f7e4bb)

4. **Not able to withdraw rewards**

   Depositors are getting 2 different rewards: HBAR and SAUCE tokens. If there is not enough of either of these two tokens, the user may just want to withdraw the existing one and wait for the other. However, the withdraw function only allows both of them to be withdrawn together and otherwise does not allow the execution of the transaction.

   If any of the tokens are insufficient, the transaction will be reverted by the *'safeTransferToken' function* on lines 357,394(in *safeSauceTransfer* function), 361 and 398, and the user will not receive both rewards.

   File: ./contracts/MasterChef

   Contract: MasterChef

   Functions: withdraw, deposit

   Recommendation: Create two separate functions to withdraw only the rewards, such as *withdrawSauceReward* and *withdrawHbarReward*.

   Status: Mitigated (with customer notice) (Revised commit: 78531f3a4de8d9de13bc7c614f67e62299f7e4bb)

Low

1. **Style guide violation**

   Contracts do not follow the Solidity code style guide.

Contracts: MasterChef.sol, Ownable.sol, SafeHederaTokenService.sol, HederaTokenService.sol

Recommendation: Follow the official Solidity code style guide. Pay attention to the next topics: “Order of Functions,” “Order of Layout," “Maximum Line Length,” and “Indentation.”

Status: Reported (Revised commit: 78531f3a4de8d9de13bc7c614f67e62299f7e4bb)

2. **Inconsistent code style**

Contracts use a mix of safemath and regular math operations.

Contracts: MasterChef.sol, SpookyIFO.sol

Recommendation: Use one of those approaches.

Status: Fixed (Revised commit: 78531f3a4de8d9de13bc7c614f67e62299f7e4bb)

3. **Unused variables**

Only SUCCESS, UNKNOWN constants were used. Others were unused, making the code overwhelmed and less readable.

Contract: HederaResponseCodes.sol

Recommendation: Review and clean up the code.

Status: Fixed (Revised commit: 78531f3a4de8d9de13bc7c614f67e62299f7e4bb)

4. **Functions that can be declared as external**

To save Gas, public functions that are never called in the contract should be declared as *external*.

Contract: MasterChef.sol

Functions: deposit, withdraw, emergencyWithdraw, setDevAddr, setRentPayer

Recommendation: Use the *external* attribute for functions never called from the contract.

Status: Fixed (Revised commit: 78531f3a4de8d9de13bc7c614f67e62299f7e4bb)

5. **Copy-pasting of well-known contracts**

It is better to import well-known contracts from the initial source, for example, from the OpenZeppelin repository. These contracts are in development, so importing them from open libraries will make code more flexible.

Contracts: Ownable.sol, ReentrancyGuard.sol

Recommendation: Change local imports to imports from the initial source.

Status: Reported (Revised commit: 78531f3a4de8d9de13bc7c614f67e62299f7e4bb)

6. Missing Zero Address Validation

There are no checks against the zero address on the address parameter of the specified functions.

This can lead to unwanted external calls to 0x0.

Contract: MasterChef.sol

Functions: setDevAddr, setRentPayer, setSauceAddress

Recommendation: Implement zero address checks.

Status: Fixed (Revised commit: 78531f3a4de8d9de13bc7c614f67e62299f7e4bb)

7. Unused constant

TINY_PARTS_PER_WHOLE is defined but never used.

Contract: HederaTokenService.sol

Function: global declaration

Recommendation: Remove this constant.

Status: Fixed (Revised commit: 78531f3a4de8d9de13bc7c614f67e62299f7e4bb)

8. Using experimental ABI encoder

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

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

Contracts: MasterChef.sol, HederaTokenService.sol, IHederaTokenService.sol, SafeHederaTokenService.sol

Recommendation: Remove the experimental ABI encoder.

Status: Fixed (Revised commit: 78531f3a4de8d9de13bc7c614f67e62299f7e4bb)

9. Unused modifier

costsTinycents is defined but never used.

Contract: HederaTokenService.sol

Recommendation: Remove this modifier.

Status: Fixed (Revised commit: 78531f3a4de8d9de13bc7c614f67e62299f7e4bb)

10. Redundant if statement

The checking *if (msg.value > 0)* is redundant because it has already been checked in the require statement above.

Contract: MasterChef.sol

Function: deposit

Recommendation: Remove the redundant *if* statement.

Status: Fixed (Revised commit: 78531f3a4de8d9de13bc7c614f67e62299f7e4bb)

11. Documentation issue

Provided documentation contains files not presented in the scope of Audit. *MasterChef.sol* contains methods not presented in the audited commit.

Contract: MasterChef.sol

Function: deposit

Recommendation: Review and fix issues or provide the correct documentation.

Status: Fixed (Revised commit: 78531f3a4de8d9de13bc7c614f67e62299f7e4bb)

12. Missing Zero Address Validation

There is no check against the zero address on the address parameter of the specified function.

This can lead to unwanted external calls to 0x0.

Contract: MasterChef.sol

Function: setSauceAddress

Recommendation: Implement zero address check.

Status: Fixed (Revised commit: d5e194537c4508574fa608140859e339a198a43c)

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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.