SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: SaucerSwap

Date: November 25th, 2022

This report 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 can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for SaucerSwap
Approved By	Noah Jelich | Lead Solidity SC Head at Hacken OU
Type	Swapping, Payment managing
Platform	Hedera
Network	Hedera Network
Language	Solidity
Methodology	Link
Website	https://www.saucerswap.finance/
Changelog	24.10.2022 - Initial Review 25.11.2022 - 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/saucerswaplabs/saucerswap-saucer
Commit	07e68a288b75b4cb336fd17fc36d62b717d031bd
Docs/Whitepaper	https://docs.saucerswap.finance/
Docs/Functional	-
Docs/Technical	-
Contracts Addresses	-
Contracts	File: ./contracts/BrewSaucer.sol SHA3: 39278a26330be8ad9a8254c9e466cb0cba0a93d5633e1f774b32c59ca47955da File: ./contracts/hedera/HederaResponseCodes.sol SHA3: 620251501964702a1416fae08a5d5f83ce2fae4262ddb838d3274d25c303a619 File: ./contracts/hedera/HederaTokenService.sol SHA3: 3ec43eaa6e071fac89ce5110d48206bea13a88b5bcee072977f16a8cfbdb5051 File: ./contracts/hedera/IExchangeRate.sol SHA3: 897c29fa822197ea10c75727573ee821c12184e9028095d9c2fd30949a760a51 File: ./contracts/hedera/IHederaTokenService.sol SHA3: caca5493eb23786977608091738716059211fdbcbc1c756129c0f079d77cf767 File: ./contracts/hedera/PriceOracle.sol SHA3: 9630da5c390efd9176ca176d8a7d4c98d6a2dcaf9e1e0fe50a934f56514c4748 File: ./contracts/hedera/SafeHederaTokenService.sol SHA3: 429da20c80dee47ab406c268b731a625035b57d79d8317116eca4128f42d53a6 File: ./contracts/interfaces/IPaymentSplitter.sol SHA3: 504f924df328a2089aa30f94c3f9c9319ae0b2b205a7204935b4f5c73244459d

	File: ./contracts/interfaces/ISwapper.sol SHA3: 797f568e9b1217a6db6e6881942ec98cd6105f6527ca08a881cdfd0a091455e2 File: ./contracts/interfaces/IUniswapV2Factory.sol SHA3: 3cf2d58f410b3b25081697738e8dd9fd75ef46848c56f0700a54a1843e7e72ef File: ./contracts/interfaces/IUniswapV2Pair.sol SHA3: d9d65c0b2833065a4af975ba26dc175f9e5a52064a35fb82bffc61ff46a2c191 File: ./contracts/interfaces/IWHBAR.sol SHA3: 1411e6c44ab8ef2158191cd72da10b59ad9f25e17622625c48d3463315876db7 File: ./contracts/libraries/Bits.sol SHA3: 1d27945c9c71aa7d4a5a1d2869a55d1af102174e07eedb6ee409af056e33a0ad File: ./contracts/Migrations.sol SHA3: f38ad4185f0fa410f3427a0bae9195f29bf1c8806f1a019cc727d7c39b53811d File: ./contracts/MotherShip.sol SHA3: 42c00d2e2414db83c46e3ff5e12a63d21362bf8e28ff0f32a7be36c5fb052564 File: ./contracts/PaymentSplitter.sol SHA3: 57514e325b2de4e3be0a11ce69777321d9e090c4aaf7b335b20b221aa969ecc1 File: ./contracts/PaymentSplitterHbar.sol SHA3: d280582e4df200abcb2000d19992fa3127a7b3397d7b45e9525da8a394512bd0 File: ./contracts/Swapper.sol SHA3: 3d5e20fedd4f4e5931bda8d90429a771d492fe40504cf5badad6e206b0c95174

Second review scope

Repository	https://github.com/saucerswaplabs/saucerswap-saucer
Commit	fea69ce49b94d7aa25ba4d2dc0a2c71599e048c5
Docs/Whitepaper	https://docs.saucerswap.finance/
Docs/Functional	-
Docs/Technical	-
Contracts Addresses	-
Contracts	File: ./contracts/BrewSaucer.sol

	SHA3: 6646c98a077c8747881f801b384f7e2a785107ac7cf990aac610dd44910d0d2a File: ./contracts/hedera/HederaResponseCodes.sol SHA3: 620251501964702a1416fae08a5d5f83ce2fae4262ddb838d3274d25c303a619 File: ./contracts/hedera/HederaTokenService.sol SHA3: 3ec43eaa6e071fac89ce5110d48206bea13a88b5bcee072977f16a8cfbdb5051 File: ./contracts/hedera/IExchangeRate.sol SHA3: 897c29fa822197ea10c75727573ee821c12184e9028095d9c2fd30949a760a51 File: ./contracts/hedera/IHederaTokenService.sol SHA3: caca5493eb237869776080917387160592 11fdbcbc1c756129c0f079d77cf767 File: ./contracts/hedera/PriceOracle.sol SHA3: 9630da5c390efd9176ca176d8a7d4c98d6a2dcaf9e1e0fe50a934f56514c4748 File: ./contracts/hedera/SafeHederaTokenService.sol SHA3: 429da20c80dee47ab406c268b731a625035b57d79d8317116eca4128f42d53a6 File: ./contracts/interfaces/IPaymentSplitter.sol SHA3: 27cc7d6396f005b319ba93d934d49db7d0fbd7bf9d658c7fb34500a7d22ca8f8 File: ./contracts/interfaces/ISwapper.sol SHA3: 0ce97b023dab800b3708bedb48b5d77c30995257ca9999ed86b6db555ec23d56 File: ./contracts/interfaces/IUniswapV2Factory.sol SHA3: 3cf2d58f410b3b25081697738e8dd9fd75ef46848c56f0700a54a1843e7e72ef File: ./contracts/interfaces/IUniswapV2Pair.sol SHA3: d9d65c0b2833065a4af975ba26dc175f9e5a52064a35fb82bffc61ff46a2c191 File: ./contracts/interfaces/IWHBAR.sol SHA3: 1411e6c44ab8ef2158191cd72da10b59ad9f25e17622625c48d3463315876db7 File: ./contracts/libraries/Bits.sol SHA3: 1d27945c9c71aa7d4a5a1d2869a55d1af102174e07eedb6ee409af056e33a0ad File: ./contracts/Migrations.sol SHA3: f38ad4185f0fa410f3427a0bae9195f29bf1c8806f1a019cc727d7c39b53811d File: ./contracts/MotherShip.sol SHA3: 412e248717ec42b964d73ad3b0aed6ae593cfbc9da6dd8cc5a588147f672a5bb

	File: ./contracts/PaymentSplitter.sol SHA3: cf7bf36b78837298d4b254995209d426351aa092686ddfa6680246555853aca5 File: ./contracts/PaymentSplitterHbar.sol SHA3: ad25e077f16c7a8e0bd2005af7a488d85fcce8a4810d50e1592b6e5c54d5fba7 File: ./contracts/Swapper.sol SHA3: 61de195edb6d1320eda00f245b14e8837c09f9602e05d3cf0ddb43ba5c4e1004

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 scoring methodology.

Documentation quality

The total Documentation Quality score is **9** out of **10**.

- Functional requirements are provided.
- Technical description that demonstrates deployment instructions, instructions on how to run tests etc, is provided.
- Function explanation as NatSpec format is mostly followed in the code.

Code quality

The total Code Quality score is **9** out of **10**.

- The development environment is configured.
- Code architecture is well-designed.
- Code mostly follows the style guide.
- Unused declarations are detected.

Test coverage

Test coverage of the project is **46%** (function coverage).

- Since coverage could not be run, the percentage of test coverage could not be measured.
- Hacken made a custom tool (attached with the report) for approximating the functional coverage statically.

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: **9.7**.

1
2
3
4
5
6
7
8
9
10
The final score

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical

24 October 2022	9	1	3	0
16 November 2022	2	0	0	0
28 November 2022	2	0	0	0

Checked Items

We have audited the Customers' smart contracts for commonly known and more specific vulnerabilities. Here are some items 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.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
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 required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed
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 EIP-155	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery	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.	Failed
EIP standards violation	EIP	EIP standards should not be violated.	Passed
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 should not be any cases when execution fails due to the block Gas limit.	Passed
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
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
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, which may be changed in the future.	Passed

System Overview

SaucerSwap is a mixed-purpose system with the following contracts:

- BrewSaucer - handles rewards for XSAUCE holders by trading tokens collected from fees for SAUCE.
- *MotherShip* – a contract that allows swapping SAUCE tokens to XSAUCE tokens and vice versa.
- PaymentSplitter – a token payment management contract that distributes the contract balance to the shareholders. New payee addresses can be added, and existing share amounts can be set.
- PaymentSplitterHbar – a native token payment management contract that distributes the contract balance to the shareholders. New payee addresses can be added, and existing share amounts can be set.
- Swapper – a contract that receives tokens to transfer to the pair contract and calls the swap function in the pair contract using an interface.
- HederaTokenService – an abstract contract that provides main Hedera token operations such as transferring, minting, token associating. It helps to create fungible tokens.
- SafeHederaTokenService – an abstract contract that does the same operations with HederaTokenService but checks the response codes of the transactions.
- HederaResponseCodes – an abstract contract that stores the Hedera transactions' response codes.
- IExchangeRate – interface of ExchangeRate contract to convert tiny cents into tiny bars or vice versa.
- IHederaTokenService – interface of HederaTokenService contract.
- Bits – a library that sets the bit at the given 'index' in 'self' to '1'.

Privileged roles

- The owner of the *PaymentSplitter* contract can add a payee address and change an existing payee's share amount.
- The owner of the PaymentSplitterHbar contract can add a payee address and change an existing payee's share amount.
- The owner of the BrewSaucer contract can add/revoke Auth addresses, set stake address, splitter address, max burn length, max conversion length, WHbar contract address, developer cut amount and developer address.
- Auth privileged role of the BrewSaucer contract can
  - associate the tokens to BrewSaucer contract
  - set slippage
  - set bridge address
  - burn tokens
  - convert tokens
  - release sauce and hbar from respective splitters

- swap WHbar
- send SAUCE tokens to Mothership contract.

Findings

■■■■ Critical

No critical severity issues were found.

■■■ High

1. Checks-Effects-Interactions Pattern Violation

In *release* function, first, the payment amount is sent to the payee then the owed amount is updated as zero. Although the function has a reentrancy guard, a payee contract that has *receive fallback* function can easily revert the transaction after receiving the tokens and send them to another address.

This may lead payees to steal the funds.

Path: ./contracts/PaymentSplitterHbar.sol : release()

Recommendation: Edit the order of internal state changes to make it safe. First, update the owed amount, then make the payment.

Status: Fixed (Revised commit: fea69ce49b94d7aa25ba4d2dc0a2c71599e048c5)

2. Requirement Compliance

On line 30, devCut variable’s comment says it is initialized as 20% in the constructor. However, it remained zero.

Path: ./contracts/BrewSaucer.sol

Recommendation: Initialize the variable in the constructor.

Status: Fixed (Revised commit: fea69ce49b94d7aa25ba4d2dc0a2c71599e048c5)

3. Authorization Through tx.origin

Preventing calls from contracts is highly discouraged. It breaks composability, breaks support for smart wallets like Gnosis Safe, and does not provide security since it can be circumvented by calling from a contract constructor.

Path: ./contracts/BrewSaucer.sol

Recommendation: Remove the mandatory EOA check and protect contract against flashloan attacks using previous blocks data/ price data buffer.

Status: Fixed (Revised commit: fea69ce49b94d7aa25ba4d2dc0a2c71599e048c5)

■■ Medium

1. Checks-Effects-Interactions Pattern Violation

The function release returns uint. The call made to the releaseFromSplitters does not check its return value. This means that the contract will continue its execution even wrong amount of token is released.

Path: ./contracts/BrewSaucer.sol : releaseFromSplitters()

Recommendation: Implement a check of the returning value.

Status: Fixed (Revised commit: fea69ce49b94d7aa25ba4d2dc0a2c71599e048c5)

Low

1. Checks-Effects-Interactions Pattern Violation

As a best practice, always follow the safest order when a function has external calls.

Path: ./contracts/MotherShip.sol : enter()

Recommendation: First, receive SAUCE tokens from the user and then mint XSAUCE tokens to the user's address.

Status: Fixed (Revised commit: fea69ce49b94d7aa25ba4d2dc0a2c71599e048c5)

2. Unused Variables

_totalReleased and *_released* variables are never initialized and used anywhere.

Redundant declarations cause unnecessary Gas consumption and decrease code readability.

Paths: ./contracts/PaymentSplitter.sol

./contracts/PaymentSplitterHbar.sol

Recommendation: Remove the unused variables or initialize them in required places.

Status: Fixed (Revised commit: fea69ce49b94d7aa25ba4d2dc0a2c71599e048c5)

3. Commented Code Parts

Line 145 has commented code. This does not create security issues, but users may interpret it as an unfinished implementation.

Paths: ./contracts/PaymentSplitter.sol : adjustSharesPayee()

./contracts/PaymentSplitterHbar.sol : adjustSharesPayee()

Recommendation: Remove the commented code part.

Status: Fixed (Revised commit: fea69ce49b94d7aa25ba4d2dc0a2c71599e048c5)

4. Style Guide Violation

To provide consistency, all contracts should follow the official style guide.

Paths: all

Recommendation: Follow the official Solidity style guide. https://docs.soliditylang.org/en/v0.8.13/style-guide.html

Status: Reported (Revised commit: fea69ce49b94d7aa25ba4d2dc0a2c71599e048c5)

5. Non-Finalized Code

PriceOracle contract is for test purposes only. Therefore, it should be removed from the project.

Paths: all

Recommendation: Remove the *PriceOracle* contract from the project.

Status: Fixed (Revised commit: fea69ce49b94d7aa25ba4d2dc0a2c71599e048c5)

6. Variable that Should Be Declared Constant

maxNumberOfPayees variable is declared immutable, but it is initialized during declaration. Because of that, it should be constant instead of immutable.

BOUNTY_FEE variable is not changed anywhere and is declared as a variable. To save Gas, it should be declared as constant.

Paths: ./contracts/PaymentSplitter.sol

./contracts/PaymentSplitterHbar.sol

./contracts/BrewSaucer.sol

Recommendation: Change the variables' type to constant.

Status: Fixed (Revised commit: fea69ce49b94d7aa25ba4d2dc0a2c71599e048c5)

7. Functions that Can Be Declared External

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

Path: ./contracts/Mothership.sol: enter(), leave()

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

Status: Fixed (Revised commit: fea69ce49b94d7aa25ba4d2dc0a2c71599e048c5)

8. Redundant Assignment

Boolean variables take a false value as default. Therefore, there is no need to assign a false value to *anyAuth* variable during global declaration.

Path: ./contracts/BrewSaucer.sol

Recommendation: Remove the assignment to save Gas.

Status: Fixed (Revised commit: fea69ce49b94d7aa25ba4d2dc0a2c71599e048c5)

9. Redundant Code

anyAuth checks should be inside the *onlyAuth* modifier, based on its logic. So, instead of writing the require statements for each function, use only the modifier.

Path: ./contracts/BrewSaucer.sol

Recommendation: Move *anyAuth* checks into the *onlyAuth* modifier.

Status: Fixed (Revised commit: fea69ce49b94d7aa25ba4d2dc0a2c71599e048c5)

10. Unused Event

LogSetAnyAuth event is not used anywhere, although it is declared.

Redundant code consumes unnecessary Gas.

Path: ./contracts/BrewSaucer.sol

Recommendation: Remove the LogSetAnyAuth event.

Status: New (Revised commit: fea69ce49b94d7aa25ba4d2dc0a2c71599e048c5)

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 report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted to and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and 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.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

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, Consultant cannot guarantee the explicit security of the audited smart contracts.