auditace.tech

AuditAce

FROM INCEPTION TO SUCCESS

Smart Contract Audit

FOR

SoFitech

DATED : 21 October 23'

AUDIT SUMMARY

Project name – SoFitech

Date: 23 October 2023

Scope of Audit- Audit Ace was consulted to conduct the smart contract audit of the solidity source codes.

Audit Status: Passed

Issues Found

Status	Critical	High	Medium	Low	Suggestion
Open	0	0	1	2	0
Acknowledged	0	0	0	0	0
Resolved	0	0	0	0	0

USED TOOLS

Tools:

1- Manual Review:
A line by line code review has been performed by audit ace team.

2- BSC Test Network: All tests were conducted on the BSC Test network, and each test has a corresponding transaction attached to it. These tests can be found in the "Functional Tests" section of the report.

3- Slither :
The code has undergone static analysis using Slither.

Token Information

Contract Address :
0xe4220986c78Db3Cf9bd845ca08b078636D99bD7d

Name: SoFitech

Symbol: ---

Decimals: ---

Network: BaseChain

Token Type: ----

Owner: 0x7e28FCE25Dd080eF7eecF699a7DeB60ff3348bFc

Deployer: 0x7e28FCE25Dd080eF7eecF699a7DeB60ff3348bFc

Token Supply: ---

Checksum:
1666029b29a5f1ae543a23971ebc1e066fc0f1b5

Testnet version:
The tests were performed using Foundry which is a smart contract development toolchain;

AUDIT METHODOLOGY

The auditing process will follow a routine as special considerations by Auditace:

- Review of the specifications, sources, and instructions provided to Auditace to make sure the contract logic meets the intentions of the client without exposing the user's funds to risk.
- Manual review of the entire codebase by our experts, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
- Specification comparison is the process of checking whether the code does what the specifications, sources, and instructions provided to Auditace describe.
- Test coverage analysis determines whether the test cases are covering the code and how much code isexercised when we run the test cases.
- Symbolic execution is analysing a program to determine what inputs cause each part of a program to execute.
- Reviewing the codebase to improve maintainability, security, and control based on the established industry and academic practices.

VULNERABILITY CHECKLIST

- [x] Return values of low-level calls
- [x] Private modifier
- [x] Multiple Sends
- [x] Using Suicide
- [x] Gas Limitand Loops
- [x] Address hardcoded
- [x] Exception Disorder
- [x] Using inline assembly
- [x] Divide before multiply
- [x] Missing Zero Address Validation
- [x] Compiler version not fixed
- [x] **Gasless Send**
- [x] Using block.timestamp
- [x] Re-entrancy
- [x] Tautology or contradiction
- [x] Timestamp Dependence
- [x] Revert/require functions
- [x] Use of tx.origin
- [x] Integer overflow/underflow
- [x] Dangerous strict equalities
- [x] Using SHA3
- [x] Using throw

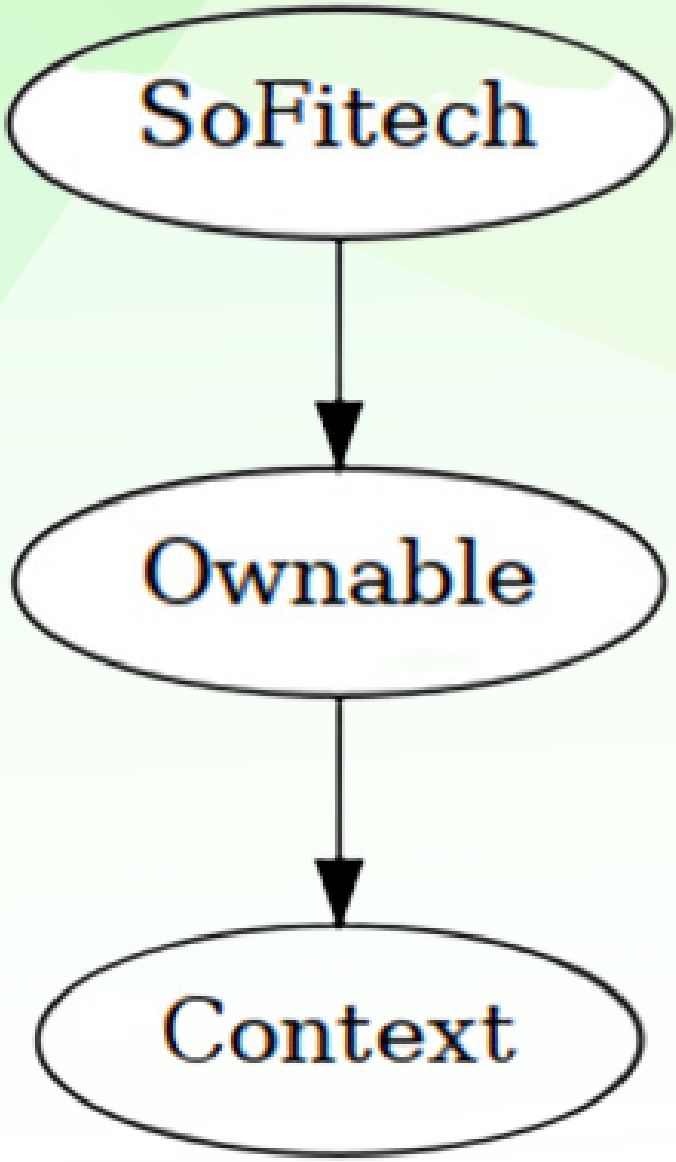
CLASSIFICATION OF RISK

Severity	Description
Critical	These vulnerabilities could be exploited easily and can lead to asset loss, data loss, asset, or data manipulation. They should be fixed right away.
High-Risk	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.
Medium-Risk	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.
Low-Risk	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.
Gas Optimization /Suggestion	A vulnerability that has an informational character but is not affecting any of the code.

Findings

Severity	Found
Critical	0
High-Risk	0
Medium-Risk	1
Low-Risk	2
Gas Optimization / Suggestions	0

INHERITANCE TREE



STATIC ANALYSIS

```
'solc --version' running
'solc ./contracts/Token.sol --combined-json abi,ast,bin,bin-runtime,srcmap,srcmap-runtime,userdoc,devdoc,hashes --allow-paths .,/home/roholah/Desktop/Deploy-BSC-Token/c
ontracts' running
INFO:Detectors:
SoFitech.getTokensOfOwner(address)._owner (contracts/Token.sol#129) shadows:
        - Ownable._owner (contracts/Token.sol#18) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
SoFitech.setProtocolFeePercent(uint256) (contracts/Token.sol#120-122) should emit an event for:
        - protocolFeePercent = _feePercent (contracts/Token.sol#121)
SoFitech.setOwnerFeePercent(uint256) (contracts/Token.sol#124-126) should emit an event for:
        - ownerFeePercent = _feePercent (contracts/Token.sol#125)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic
INFO:Detectors:
SoFitech.constructor(address,uint256,uint256)._protocolFeeDestination (contracts/Token.sol#107) lacks a zero-check on :
                - protocolFeeDestination = _protocolFeeDestination (contracts/Token.sol#111)
SoFitech.setFeeDestination(address)._feeDestination (contracts/Token.sol#116) lacks a zero-check on :
                - protocolFeeDestination = _feeDestination (contracts/Token.sol#117)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Different versions of Solidity are used:
        - Version used: ['>=0.8.2<0.9.0', '^0.8.0']
        - >=0.8.2<0.9.0 (contracts/Token.sol#62)
        - ^0.8.0 (contracts/Token.sol#5)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
Context._msgData() (contracts/Token.sol#12-14) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.0 (contracts/Token.sol#5) allows old versions
Pragma version>=0.8.2<0.9.0 (contracts/Token.sol#62) is too complex
solc-0.8.17 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in SoFitech.buyShares(string,uint256) (contracts/Token.sol#166-205):
        - (successProtocolFee) = address(protocolFeeDestination).call{value: protocolFee}() (contracts/Token.sol#196-198)
        - (successOwnerFee) = address(currentTokenOwner).call{value: ownerFee}() (contracts/Token.sol#201-203)
Low level call in SoFitech.sellShares(string,uint256) (contracts/Token.sol#207-250):
        - (successUser) = address(msg.sender).call{value: price - protocolFee - ownerFee}() (contracts/Token.sol#235-237)
        - (successProtocolFee) = address(protocolFeeDestination).call{value: protocolFee}() (contracts/Token.sol#239-241)
        - (successOwnerFee) = address(currentTokenOwner).call{value: ownerFee}() (contracts/Token.sol#243-245)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter SoFitech.setFeeDestination(address)._feeDestination (contracts/Token.sol#116) is not in mixedCase
Parameter SoFitech.setProtocolFeePercent(uint256)._feePercent (contracts/Token.sol#120) is not in mixedCase
Parameter SoFitech.setOwnerFeePercent(uint256)._feePercent (contracts/Token.sol#124) is not in mixedCase
Parameter SoFitech.getTokensOfOwner(address)._owner (contracts/Token.sol#129) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Slither:./contracts/Token.sol analyzed (3 contracts with 88 detectors), 16 result(s) found
```

Result => A static analysis of contract's source code has been performed using slither,

No major issues were found in the output

CONTRACT ASSESMENT

Contract	Type	Bases		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation		
└	_msgSender	Internal 🔒		
└	_msgData	Internal 🔒		
Ownable	Implementation	Context		
└	<Constructor>	Public ❗	🛑	NO ❗
└	owner	Public ❗		NO ❗
└	_checkOwner	Internal 🔒		
└	renounceOwnership	Public ❗	🛑	onlyOwner
└	transferOwnership	Public ❗	🛑	onlyOwner
└	_transferOwnership	Internal 🔒	🛑	
SoFitech	Implementation	Ownable		
└	<Constructor>	Public ❗	🛑	NO ❗
└	setFeeDestination	Public ❗	🛑	onlyOwner
└	setProtocolFeePercent	Public ❗	🛑	onlyOwner
└	setOwnerFeePercent	Public ❗	🛑	onlyOwner
└	getTokensOfOwner	External ❗		NO ❗
└	getPrice	Public ❗		NO ❗
└	createTokenId	Public ❗	🛑	nonReentrant
└	buyShares	Public ❗	💵	nonReentrant
└	sellShares	Public ❗	🛑	nonReentrant
└	generateTokenId	Public ❗		NO ❗
└	getBuyPrice	Public ❗		NO ❗
└	getSellPrice	Public ❗		NO ❗
└	getBuyPriceAfterFee	Public ❗		NO ❗
└	getSellPriceAfterFee	Public ❗		NO ❗

Legend

Symbol	Meaning
🛑	Function can modify state
💵	Function is payable

MANUAL TESTING

Input validation – Unbounded fees
Severity: Medium
function: setProtocolFeePercent - setOwnerFeePercent
Status: Open

Overview:
the setProtocolFeePercent and setOwnerFeePercent functions are used to update protocol and owner fee percent.
Protocol fee percent (in ether) is a portion of buying/selling price which will be sent to protocol wallet.
Owner fee percent (in ether) is a portion of buying/selling price which will be sent to owner wallet. (owner of ID).
None of this functions are validating "_feePercent" to be less than 1 ether.

Impact:
Setting feePercent to more than 1 ether causes underflow when selling IDs

```
function setProtocolFeePercent(uint256 _feePercent) public onlyOwner {
    protocolFeePercent = _feePercent;
}

function setOwnerFeePercent(uint256 _feePercent) public onlyOwner {
    ownerFeePercent = _feePercent;
}
```

Suggestion
Ensure that _feePercent is less than 1 ether

```
function setProtocolFeePercent(uint256 _feePercent) public onlyOwner {
    require(_feePercent < 1 ether);
    protocolFeePercent = _feePercent;
}

function setOwnerFeePercent(uint256 _feePercent) public onlyOwner {
    require(_feePercent < 1 ether);
    ownerFeePercent = _feePercent;
```

MANUAL TESTING

Back running – Losing funds for later investors
Severity: Non-critical / Discussion
function: sellShares
Status: Open

Overview:
The current implementation of sellShares and buyShares allows for a scenario where earlier investors can potentially sell their shares at a much higher price than what they paid for, after new purchases have been made. This can particularly harm new buyers, whose shares suddenly decrease in value after previous holders sell.
Proof of concept:
- Alice creates a token with ID of “soFiX”
- Alice purchases 10 tokens of ID “soFiX” and pays 0.0336875 ether
- Bob purchases 10 tokens of ID “soFiX” and pays 0.2174375 ether
- Alice sells her 10 tokens and receives 0.12425 ether
- Price of Bob tokens drops to 0.0048125 ether
In the above example, Bob lost 0.212625 ether, and Alice gained 0.0905625 ether in profits. This is a significant disparity and could be viewed as an unfair advantage for earlier investors.

Suggestion
To address this, consider implementing a pricing model that is more balanced and doesn't allow for such large profits for earlier investors at the expense of later investors. Some options could include:
1.Time-Based Lock: Implement a time-based lock for selling shares. If a user buys shares, they should not be able to sell them until a certain period has elapsed.
2.Price Smoothing: Alter the pricing algorithm to be less sensitive to changes in supply. This way, the impact of new purchases on the sell price of existing shares would be minimized.
3.Rate Limiting: Introduce rate limits on how many shares can be sold within a specific timeframe to prevent massive dumps that would affect new buyers negatively.

MANUAL TESTING

Event Emission – Lack of event emission
Severity: Non-critical / Discussion
function: setProtocolFeePercent – setOwnerFeePercent - setFeeDestination
Status: Open

Overview:
Below functions are changing state, but not emitting any events.

```
function setFeeDestination(address _feeDestination) public onlyOwner {
    protocolFeeDestination = _feeDestination;
}

function setProtocolFeePercent(uint256 _feePercent) public onlyOwner {
    protocolFeePercent = _feePercent;
}

function setOwnerFeePercent(uint256 _feePercent) public onlyOwner {
    ownerFeePercent = _feePercent;
}
```

Suggestion
Consider emitting events from above functions.

DISCLAIMER

All the content provided in this document is for general information only and should not be used as financial advice or a reason to buy any investment. Team provides no guarantees against the sale of team tokens or the removal of liquidity by the project audited in this document. Always Do your own research and protect yourselves from being scammed. The Auditace team has audited this project for general information and only expresses their opinion based on similar projects and checks from popular diagnostic tools. Under no circumstances did Auditace receive a payment to manipulate those results or change the awarding badge that we will be adding in our website. Always Do your own research and protect yourselves from scams. This document should not be presented as a reason to buy or not buy any particular token. The Auditace team disclaims any liability for the resulting losses.

ABOUT AUDITACE

We specializes in providing thorough and reliable audits for Web3 projects. With a team of experienced professionals, we use cutting-edge technology and rigorous methodologies to evaluate the security and integrity of blockchain systems. We are committed to helping our clients ensure the safety and transparency of their digital assets and transactions.

https://auditace.tech/

https://t.me/Audit_Ace

https://twitter.com/auditace_

https://github.com/Audit-Ace