SecureWise

AUDIT REPORT

SecureWise

SAFUCORE

SAFUCORE

https://securewise.info/ https://t.me/securewisehub https://github.com/securewise

Table of Contents

Disclaimer

SecureWise provides the smart contract audit of solidity. Audit and report are for informational purposes only and not, nor should be considered, as an endorsement to engage with, invest in, participate, provide an incentive, or disapprove, criticise, discourage, or purport to provide an opinion on any particular project or team.

This audit report doesn't provide any warranty or guarantee regarding the nature of the technology analysed. These reports, in no way, provide investment advice, nor should be used as investment advice of any sort. Investors must always do their own research and manage their risk.

DISCLAIMER: By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis, and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and SecureWise and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) SecureWise owe no duty of care towards you or any other person, nor does SecureWise make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and SecureWise hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, SecureWise hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against SecureWise, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report.

Overview

Token Name: SaFuCore**(SFC)**

Methodology: Automated Analysis, Manual Code Review

Language: Solidity

Contract Address: 0x306829001CC6E5d970384DFf772f950a3FCe8Cca

ContractLink: https://scan.coredao.org/address/0x306829001CC6E5d970384DFf772f950a3FCe8Cca

Network: Core

Supply: 1.000.000

Website: https://www.safucore.com/

Twitter: https://twitter.com/SafuCore0

Telegram: https://t.me/SafuCore

Report Date: March 17, 2023

Quick Result

SecureWise has applied the automated and manual analysis of Smart Contract and were reviewed for common contract vulnerabilities and centralized exploits

⚠	The owner can set the fees with limit of 30% at max
⚠	The owner can exclude accounts from fees
⚠	The owner can set max transaction amount within reasonable limits
⚠	The owner can change max wallet token amount within reasonable limits
⚠	The owner can change swap settings

Page 12 for more details

SaFuCore(SFC) has succesfully **PASSED** the smart contract audit with **MEDIUM** and **LOW** **severity issue**

Auditing Approach and Methodologies

SecureWise has performed starting with analyzing the code, issues, code quality, and libraries. Reviewed line-by-line by our team. Finding any potential issue like race conditions, transaction-ordering dependence, timestamp dependence, and denial of service attacks.

Methodology

- Understanding the size, scope and functionality of your project's source code
- Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
- Testing and automated analysis of the Smart Contract to determine proper logic has been followed throughout the whole process
- Deploying the code on testnet using multiple live test
- Analyzing a program to determine the specific input that causes different parts of a program to execute its functions.
- Checking whether all the libraries used in the code are on the latest version.

Goals

Smart Contract System is secure, resilient and working according to the specifications and without any vulnerabilities.

Risk Classification

High: Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, of the contract and its functions. Must be fixed as soon as possible.

Medium: Bugs or issues with that may be subject to exploit, though their impact is somewhat limited. Must be fxed as soon as possible.

Low: Effects are minimal in isolation and do not pose a signifcant danger to the project or its users. Issues under this classifcation are recommended to be fixed nonetheless.

Automated Analysis

Symbol	Meaning
●	Function can modify state
💵	Function is payable

SafeMath	Library			
L	add	Internal 🔒		
L	sub	Internal 🔒		
L	mul	Internal 🔒		
L	div	Internal 🔒		
L	mod	Internal 🔒		
L	tryAdd	Internal 🔒		
L	trySub	Internal 🔒		
L	tryMul	Internal 🔒		
L	tryDiv	Internal 🔒		
L	tryMod	Internal 🔒		
L	sub	Internal 🔒		
L	div	Internal 🔒		
L	mod	Internal 🔒		
IERC20	Interface			
L	totalSupply	External !		NO !
L	decimals	External !		NO !
L	symbol	External !		NO !
L	name	External !		NO !
L	getOwner	External !		NO !
L	balanceOf	External !		NO !
L	transfer	External !	●	NO !
L	allowance	External !		NO !
L	approve	External !	●	NO !
L	transferFrom	External !	●	NO !
Ownable	Implementation			
L		Public !	●	NO !
L	isOwner	Public !		NO !
L	transferOwnership	Public !	●	onlyOwner
IFactory	Interface			
L	createPair	External !	●	NO !

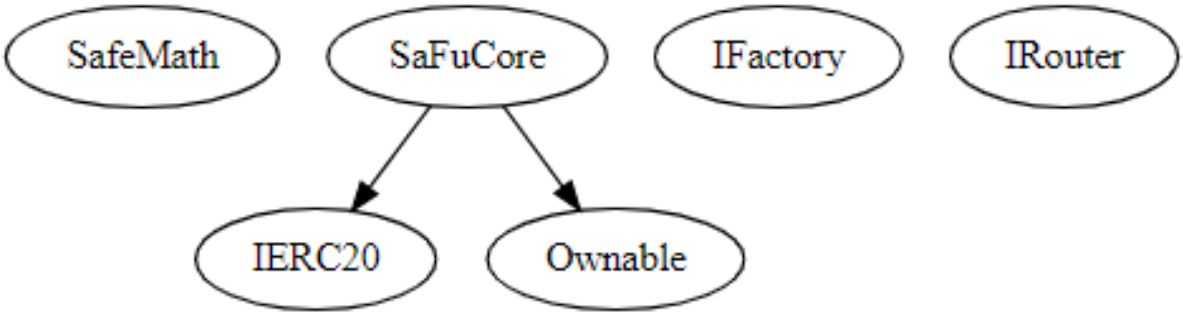
Automated Analysis

IRouter	Interface			
L	factory	External !		NO !
L	WETH	External !		NO !
L	swapExactTokensForETHSupportingFeeOnTransferTokens	External !	🔴	NO !
L	addLiquidityETH	External !	💵	NO !
SaFuCore	Implementation	IERC20, Ownable		
L		Public !	🔴	Ownable
L		External !	💵	NO !
L	name	Public !		NO !
L	symbol	Public !		NO !
L	decimals	Public !		NO !
L	getOwner	External !		NO !
L	balanceOf	Public !		NO !
L	transfer	Public !	🔴	NO !
L	allowance	Public !		NO !
L	excludeFromFees	External !	🔴	onlyOwner
L	approve	Public !	🔴	NO !
L	totalSupply	Public !		NO !
L	_maxWalletToken	Public !		NO !
L	_maxTxAmount	Public !		NO !
L	checkTx	Internal 🔒		
L	_transfer	Private 🔐	🔴	
L	setFee	External !	🔴	onlyOwner
L	setMax	External !	🔴	onlyOwner
L	setSwapThreshold	External !	🔴	onlyOwner
L	setMarketing	External !	🔴	onlyOwner
L	checkMaxWallet	Internal 🔒		
L	swapCounters	Internal 🔒	🔴	
L	addLiquidity	Private 🔐	🔴	
L	checkTxLimit	Internal 🔒		
L	swapBack	Private 🔐	🔴	lockTheSwap
L	swapTokensForETH	Private 🔐	🔴	

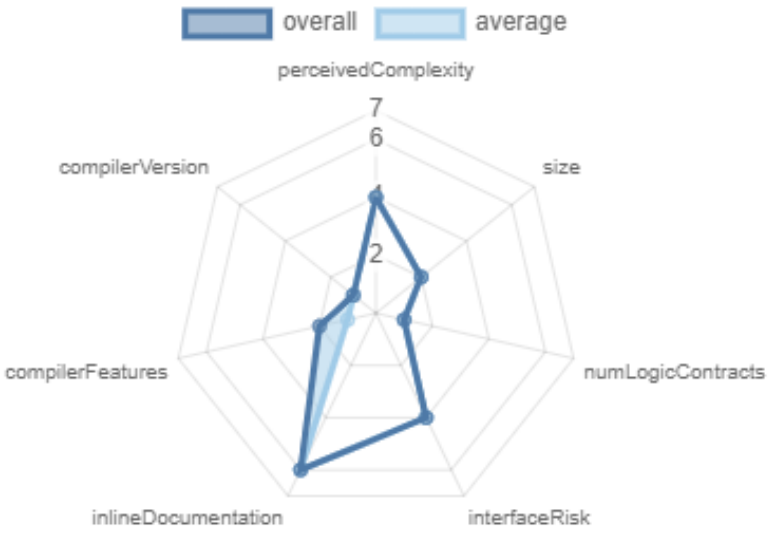
Automated Analysis

L	swapAndLiquify	Private 🔐	🔴	
L	shouldSwapBack	Internal 🔒		
L	swapBack	Internal 🔒	🔴	
L	shouldTakeFee	Internal 🔒		
L	getTotalFee	Internal 🔒		
L	takeFee	Internal 🔒	🔴	
L	transferFrom	Public ❗	🔴	NO ❗
L	_approve	Private 🔐	🔴	

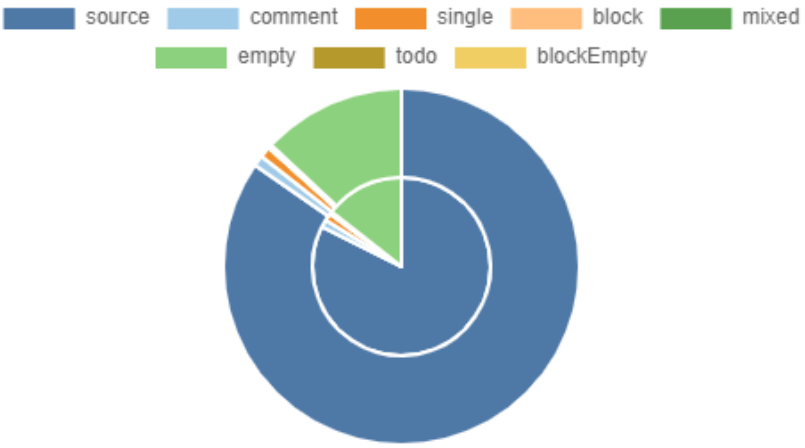
Inheritance Graph



Risk



Source Lines



Contract Summary

Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
3	3	595	427	360	6	257	
3	**3**	**595**	**427**	**360**	**6**	**257**	

Components

Contracts	Libraries	Interfaces	Abstract
1	1	3	1

Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

Public	Payable
35	2

External	Internal	Private	Pure	View
22	47	6	18	20

StateVariables

Total	Public
27	9

Capabilities

Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
0.8.16		yes	—	—

Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	ECRecover	New/Create/Create2
yes	—	—	—	—	—

TryCatch	Σ Unchecked
yes	yes

Manual Review

The owner can set the fees with limit of 30% at max

```
function setFee(
    uint256 _bLiquidFee,
    uint256 _bMarketingFee,
    uint256 _sLiquidFee,
    uint256 _sMarketingFee
) external onlyOwner {
    buyFee = _bLiquidFee.add(_bMarketingFee);
    sellFee = _sLiquidFee.add(_sMarketingFee);
    require(buyFee <= 30 && sellFee <= 30, "Must keep fees at 30% or less");
}
```

The owner can update buy and sell fees but max 30% both way. Team should be carefull management of private key of the owner's account. Updating 30% fee is a medium risk.

Recommendation

You should careffuly manage the private key of the owner's account. You should use powerful security mechanism that will prevent a single user from accessing the contract owner functions. That risk can be prevented by temporarily locking the contract or renouncing ownership

The owner can exclude accounts from fees

```
function excludeFromFees(address _address, bool _enabled)
    external
    onlyOwner
{
    isExcludedFromFees[_address] = _enabled;
}
```

Authorizing privileged roles to exclude accounts from fees. These cause can affect decentralization. After excluding the user from accounts, the user trades without paying a any fee and the other user sees it). But may apply in some cases like (owner wallets, contract...)

Recommendation

You should careffuly manage the private key of the owner's account. You should use powerful security mechanism that will prevent a single user from accessing the contract owner functions. That risk can be prevented by temporarily locking the contract or renouncing ownership

Manual Review

There is a transaction timelock limits but it can not changed

```
2 references | Control flow graph
function _transfer(
    address sender,
    address recipient,
    uint256 amount
) private {
    if (startBlock == 0 && recipient == pair) {
        starting = true;
        startBlock = block.number;
    } else if (starting == true && block.number > (startBlock + 3)) {
        starting = false;
    }
```

There is a transaction limit for prevending frontrun bots and its 3 second. It can not change from the owner

The owner can set max transaction amount within reasonable limits

```
function setMax(uint256 _tx, uint256 _wallet) external onlyOwner {
    uint256 newTx = (totalSupply() * _tx) / 10000;
    uint256 newWallet = (totalSupply() * _wallet) / 10000;
    uint256 limit = totalSupply().mul(5).div(1000);
    require(
        newTx >= limit && newWallet >= limit,
        "Max TXs and Max Wallet cannot be less than .5%"
    );
    _maxTxAmountPercent = _tx;
    _maxWalletPercent = _wallet;
}
```

The owner can set maximum transaction within reasonable limits.
The owner can set max transaction limit 0.5% of TotalSupply.

Manual Review

The owner can change max wallet token amount within reasonable limits

```
function setMax(uint256 _tx, uint256 _wallet) external onlyOwner {
    uint256 newTx = (totalSupply() * _tx) / 10000;
    uint256 newWallet = (totalSupply() * _wallet) / 10000;
    uint256 limit = totalSupply().mul(5).div(1000);
    require(
        newTx >= limit && newWallet >= limit,
        "Max TXs and Max Wallet cannot be less than .5%"
    );
    _maxTxAmountPercent = _tx;
    _maxWalletPercent = _wallet;
}
```

The owner can set maximum transaction within reasonable limits.
The owner can set max wallet limit 0.5% of TotalSupply.

The owner can change swap settings

```
function setSwapThreshold(uint256 _amount) external onlyOwner {
    swapThreshold = _amount;
}
```

The variable **setSwapThreshold** sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then contract will swap a huge amount. This means that the value of price volatility.

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single tx. You should check maximum amount should be less than a fixed percantage of the total supply.

Manual Review

Lacks a zero-check on set wallets function

```
function setMarketing(address _marketing) external onlyOwner {
  marketing = _marketing;
}
```

Zero-address checks as input validation on address parameters is always a best practice. This is especially true for critical addresses that are immutable and set in the constructor because they cannot be changed later. Accidentally using zero addresses here will lead to failing logic or force contract redeployment and increased gas costs.

Recommendation

Add zero-address input validation for these addresses.

Allowance() currently poses a risk of a race condition

```
function allowance(
  address owner,
  address spender
) public view override returns (uint256) {
  return _allowances[owner][spender];
}
```

Recommendation

To prevent a possible race condition we recommend introducing increaseAllowance() and decreaseAllowance()

Manual Review

Access Modifiers Vulnerabilities

```
transferOwnership()
name()
symbol()
decimals()
allowance()
totalSupply()
transferFrom()
approve()
```

These functions are used as external instead of public.

Recommendation

Access control identifiers must be authenticated and set adequately to avoid possible vulnerabilities

Out date compiler version

```
pragma solidity 0.8.16;
```

Compiler is set an outdated version.

Recommendation

Set and use new versions

SecureWise

AUDIT REPORT

SecureWise

https://securewise.info/ https://t.me/securewisehub https://github.com/securewise