SecureWise

AUDIT REPORT

SecureWise

ZK CAPITAL

ZK CAPITAL

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: ZkCapital **(ZKC)**

Methodology: Automated Analysis, Manual Code Review

Language: Solidity

Contract Address: 0x7060610F4619A03584aAc702f8fFD7EDd15C833b

ContractLink: https://explorer.zksync.io/address/0x7060610F4619A03584aAc702f8fFD7EDd15C833b

Network: zkSync Era

Supply: 50000000000

Website: -

Twitter: https://twitter.com/zk_capital0?t=a1X26gF9XYfaI52xGUSihg&s=09

Telegram: https://discord.com/invite/zk-capital

Report Date: April 14, 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 fees up to 100%
	Auto liquidity is going to an externally owned account
	Owner can swap and withdraw all of the collected taxes to treasury account
	Rebase() function logic issues
	The owner can update Lp and Pair address
	The owner can exclude accounts from fees
	The owner can stop auto rebasing

Page 10 for more details

ZkCapital (ZKC) has succesfully **PASSED** the smart contract audit with **HIGH** **MEDIUM** **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.

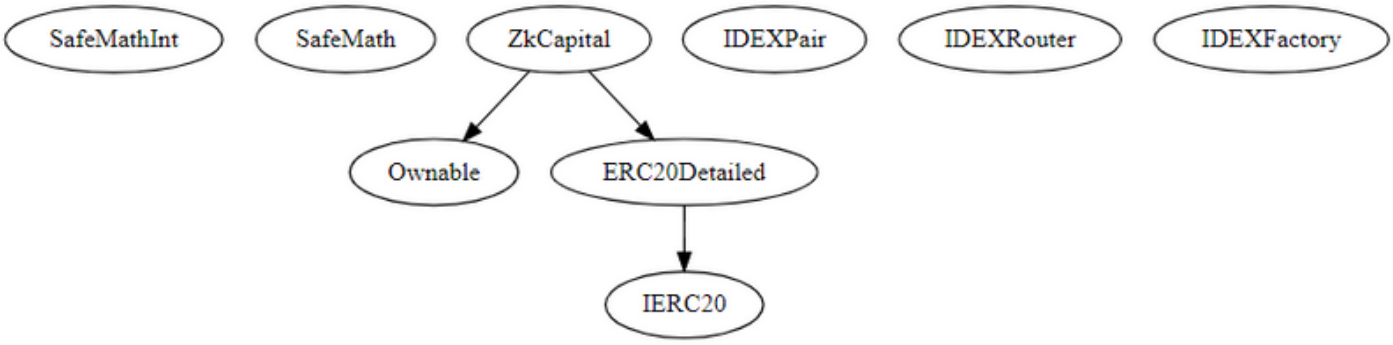
Low: Effects are minimal in isolation and do not pose a signifcant danger to the project or its users. Issues under this classifcation are recommended to be fixed nonetheless.

Automated Analysis

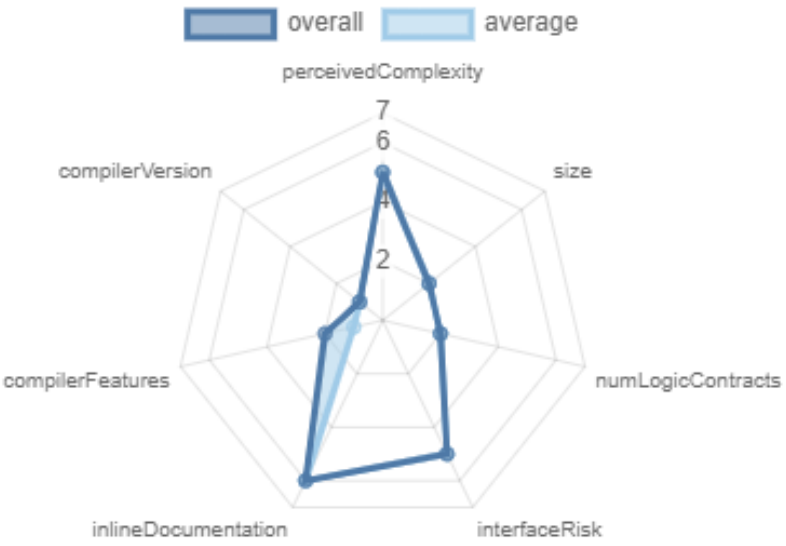
Symbol	Meaning
●	Function can modify state
💵	Function is payable

ZkCapital	Implementation	ERC20Detailed, Ownable		
L		Public !	●	ERC20Detailed Ownable
L	rebase	Internal 🔒	●	
L	transfer	External !	●	validRecipient
L	transferFrom	External !	●	validRecipient
L	_basicTransfer	Internal 🔒	●	
L	_transferFrom	Internal 🔒	●	
L	takeFee	Internal 🔒	●	
L	addLiquidity	Internal 🔒	●	swapping
L	swapBack	Internal 🔒	●	swapping
L	withdrawAllToTreasury	External !	●	swapping onlyOwner
L	shouldTakeFee	Internal 🔒		
L	shouldRebase	Internal 🔒		
L	shouldAddLiquidity	Internal 🔒		
L	shouldSwapBack	Internal 🔒		
L	setAutoRebase	External !	●	onlyOwner
L	setAutoAddLiquidity	External !	●	onlyOwner
L	allowance	External !		NO !
L	decreaseAllowance	External !	●	NO !
L	increaseAllowance	External !	●	NO !
L	approve	External !	●	NO !
L	checkFeeExempt	External !		NO !
L	getCirculatingSupply	Public !		NO !
L	isNotInSwap	External !		NO !
L	manualSync	External !	●	NO !
L	setFees	External !	●	onlyOwner
L	setFeeReceivers	External !	●	onlyOwner
L	getLiquidityBacking	Public !		NO !
L	setWhitelist	External !	●	onlyOwner
L	setPairAddress	Public !	●	onlyOwner
L	setLP	External !	●	onlyOwner
L	totalSupply	External !		NO !
L	balanceOf	External !		NO !
L	isContract	Internal 🔒		
L		External !	💵	NO !

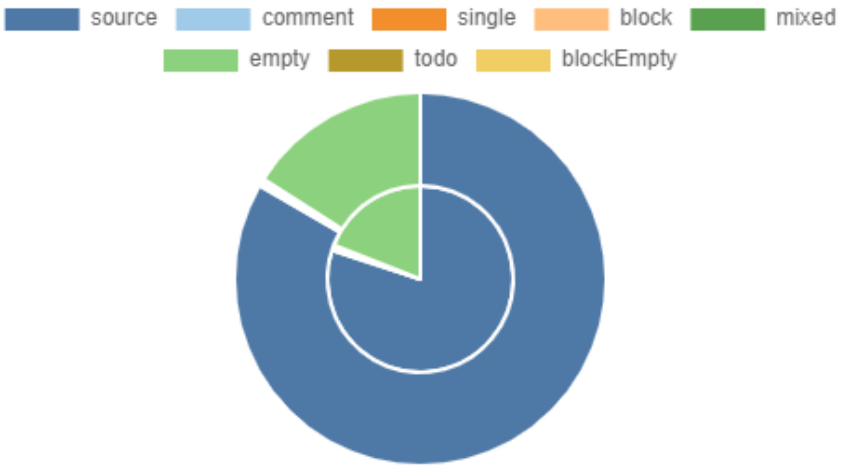
Inheritance Graph



Risk



Source Lines



Contract Summary

Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
5	4	904	632	508	3	495	
5	**4**	**904**	**632**	**508**	**3**	**495**	

Components

Contracts	Libraries	Interfaces	Abstract
2	2	4	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
94	5

External	Internal	Private	Pure	View
84	79	0	22	39

StateVariables

Total	Public
44	19

Capabilities

Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
^0.7.4		yes	yes (1 asm blocks)	

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

TryCatch	Σ Unchecked

Manual Review

The owner can set fees up to 100%

```
function setFees(uint256 _liquidityFee1,uint256 _safehouseFee1,uint256 _treasuryFee1) external onlyOwner {
    liquidityFee = _liquidityFee1;
    safehouseFee = _safehouseFee1;
    treasuryFee = _treasuryFee1;
}
```

This can be harmful because it could allow the contract owner to set fees that are excessively high, potentially leading to a situation where users are discouraged from using the contract.

Recommendation

It would be prudent to thoroughly review the contract's fee structure and determine whether there are any potential security risks. The contract owner's control over the fees should be carefully considered and potentially limited to prevent abuse or centralization.

Auto liquidity is going to an externally owned account

```
        if (amountToLiquify > 0 &&amountETHLiquidity > 0) {
            router.addLiquidityETH{value: amountETHLiquidity}(
                address(this),
                amountToLiquify,
                0,
                0,
                autoLiquidityReceiver,
                block.timestamp
            );
        }
        _lastAddLiquidityTime = block.timestamp;
    }
```

autoLiquidityReceiver account is owned and managed by the contract owner, it could raise concerns about centralization and trust in the contract, as the contract owner would have complete control over the liquidity added to the account.

Recommendation

Authorizing privileged roles to externally-owned-account (EOA) is dangerous. Send LP tokens to dead address or unreachable address.

Manual Review

Owner can swap and withdraw all of the collected taxes to treasury account

```
function withdrawAllToTreasury() external swapping onlyOwner {

    uint256 amountToSwap = _gonBalances[address(this)].div(_gonsPerFragment);
    require( amountToSwap > 0,"There is no ZkCapital token deposited in token contract");
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = router.WETH();
    router.swapExactTokensForETHSupportingFeeOnTransferTokens(
        amountToSwap,
        0,
        path,
        treasuryReceiver,
        block.timestamp
    );
}
```

This feature may be intended to benefit the project or token holders, it also presents a risk of centralization and potential misuse of funds and if the treasury account is not carefully managed or secured, it could be vulnerable to attacks that could result in the loss of funds.

Recommendation

It is recommended to thoroughly review the treasury account and determine whether it is owned and managed by the contract owner or by a trusted third party. It may also be useful to consider implementing additional safeguards or controls. also recommended to consider implementing a timelock or multi-signature requirement for any withdrawals to the treasury account

Rebase() function logic issues

```
if (deltaTimeFromInit < (365 days)) {
    rebaseRate = 2355;
} else if (deltaTimeFromInit >= (365 days)) {
    rebaseRate = 211;
} else if (deltaTimeFromInit >= ((15 * 365 days) / 10)) {
    rebaseRate = 14;
} else if (deltaTimeFromInit >= (7 * 365 days)) {
    rebaseRate = 2;
}
```

If statements might never be reached. If the second statement is reached (>= 365 days), then the other two below will not be called upon.

Recommendation

If statement is checked in order of decreasing threshold, and the appropriate rebaseRate is assigned based on the amount of time that has passed since the last rebase.

Manual Review

The owner can update Lp and Pair address

```
function setPairAddress(address _pairAddress) public onlyOwner {
    pairAddress = _pairAddress;
}

ftrace | funcSig
function setLP(address _address) external onlyOwner {
    pairContract = IDEXPair(_address);
}
```

If these functions can be called by any external address, then there could be a security issue if a malicious actor were to set a fake or compromised pair address, which could potentially lead to funds being stolen or manipulated.

Recommendation

To improve security, it's recommended to implement additional safeguards such as access control checks and verifications to ensure that only trusted addresses can call these functions. Additionally, it may be useful to consider using timelocks or multisig wallets to add an extra layer of security to these critical functions.

The owner can update safehouse address and can withdraw collected fees

```
    _gonBalances[safehouse] = _gonBalances[safehouse].add(
        gonAmount.div(feeDenominator).mul(safehouseFee)
    );
```

```
function setFeeReceivers(address _autoLiquidityReceiver,address _treasuryReceiver,address _ZKCInsuranceFundReceiver,address _safehouse
) external onlyOwner {
    autoLiquidityReceiver = _autoLiquidityReceiver;
    treasuryReceiver = _treasuryReceiver;
    ZKCInsuranceFundReceiver = _ZKCInsuranceFundReceiver;
    safehouse = _safehouse;
}
```

The safehouse is one of the fee receivers currently used for dead address, and its address can be updated by the owner using the setFeeReceivers function. If the safehouse address is updated by someone other than the owner with a malicious address, then the owner could potentially lose control of the fees meant for the safehouse. The malicious address could redirect the fees to their own account instead of the intended safehouse account, resulting in a loss of funds for the owner. Therefore, it's important to ensure that only the owner or a trusted party can update the safehouse address.

Manual Review

The owner can exclude accounts from fees

```
ftrace | funcSig
function setWhitelist(address _addr) external onlyOwner {
    _isFeeExempt[_addr] = true;
}
```

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

The owner can stop auto rebasing

```
function setAutoRebase(bool _flag) external onlyOwner {
    if (_flag) {
        _autoRebase = _flag;
        _lastRebasedTime = block.timestamp;
    } else {
        _autoRebase = _flag;
    }
}
```

By turning off auto-rebasing, the contract owner could potentially manipulate the token supply and price, which could harm the value of the token and the interests of other users.

Recommendation

It is recommended to carefully evaluate the need for this feature and assess the risks associated with giving the contract owner control over it. If the feature is deemed necessary, it is recommended to implement additional security measures to prevent abuse and manipulation, such as requiring a time lock or multi-signature control for turning the feature off.

SecureWise

AUDIT REPORT

SecureWise

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