SOLIDProof
Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY

Elvis

Audit

Security Assessment
15. March, 2022

For

Disclaimer

SolidProof.io reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team. SolidProof.io do not cover testing or auditing the integration with external contract or services (such as Unicrypt, Uniswap, PancakeSwap etc’...)

SolidProof.io Audits do not provide any warranty or guarantee regarding the absolute bug- free nature of the technology analyzed, nor do they provide any indication of the technology proprietors. SolidProof Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

SolidProof.io Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Blockchain technology and cryptographic assets present a high level of ongoing risk. SolidProof’s position is that each company and individual are responsible for their own due diligence and continuous security. SolidProof in no way claims any guarantee of security or functionality of the technology we agree to analyze.

Version	Date	Description
1.0	15. March 2022	• Layout project • Automated- /Manual-Security Testing • Summary

Network
Binance Smart Chain (BEP20)

Website
https://elvis.fit/

Telegram
https://t.me/Elvis_global_community

Twitter
https://twitter.com/ElvisToken

Instagram
https://instagram.com/overmoontoken?utm_medium=copy_link

Description

TBA

Project Engagement

During the 11th of March 2022, **Elvis Team** engaged Solidproof.io to audit smart contracts that they created. The engagement was technical in nature and focused on identifying security flaws in the design and implementation of the contracts. They provided Solidproof.io with access to their code repository and whitepaper.

Logo

Contract Link

v1.0

- https://bscscan.com/address/0xb1cF34DeCbB4B8c8D1D024b1d7b15034D2acaae0#code

Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 - 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon aspossible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as there were discovered.

Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
    i) Review of the specifications, sources, and instructions provided to SolidProof to make sure we understand the size, scope, and functionality of the smart contract.
    ii) Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    iii) Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to SolidProof describe.

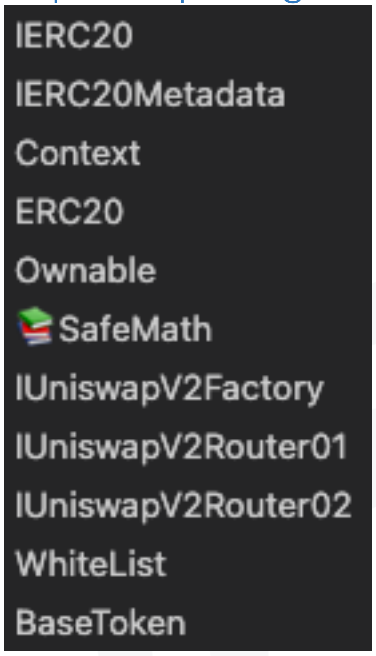
2. Testing and automated analysis that includes the following:
    i) Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    ii) Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.

3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.

4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

Used Code from other Frameworks/Smart Contracts (direct imports)

Imported packages:



IERC20
IERC20Metadata
Context
ERC20
Ownable
SafeMath
IUniswapV2Factory
IUniswapV2Router01
IUniswapV2Router02
WhiteList
BaseToken

Tested Contract Files

This audit covered the following files listed below with a SHA-1 Hash.

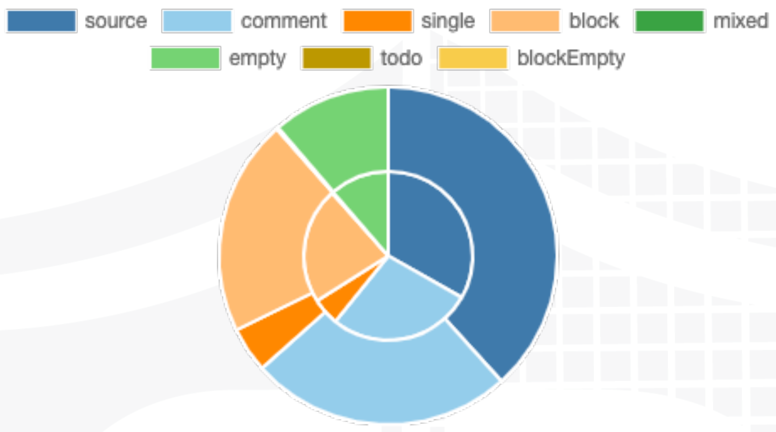
A file with a different Hash has been modified, intentionally or otherwise, after the security review. A different Hash could be (but not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of this review.

v1.0

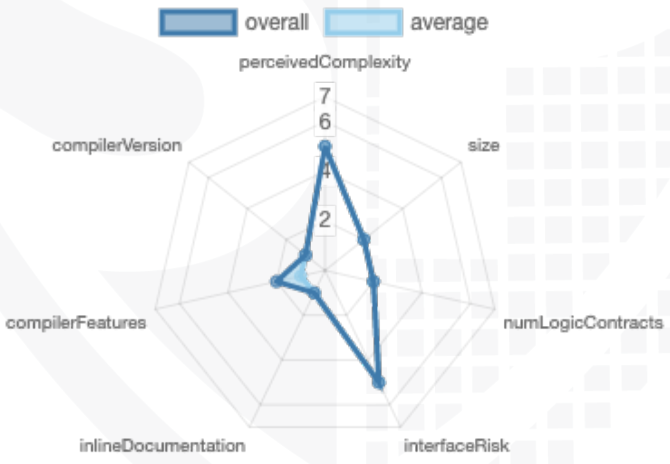
File Name	SHA-1 Hash
contracts/elvis.sol	f23107becb45a92ae843ccc4de060865fe62368b

Metrics

Source Lines
v1.0



Risk Level
v1.0



Capabilities

Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	2	1	5	4

Exposed Functions

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

Version	Public	Payable
1.0	75	6

Version	External	Internal	Private	Pure	View
1.0	45	78	6	18	29

State Variables

Version	Total	Public
1.0	24	10

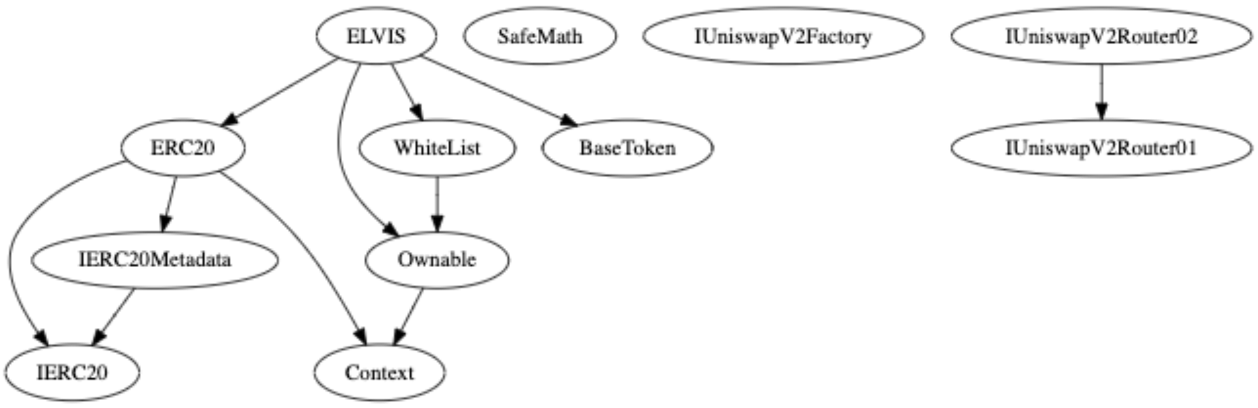
Capabilities

Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
1.0	`>=0.8.4`		`yes`		

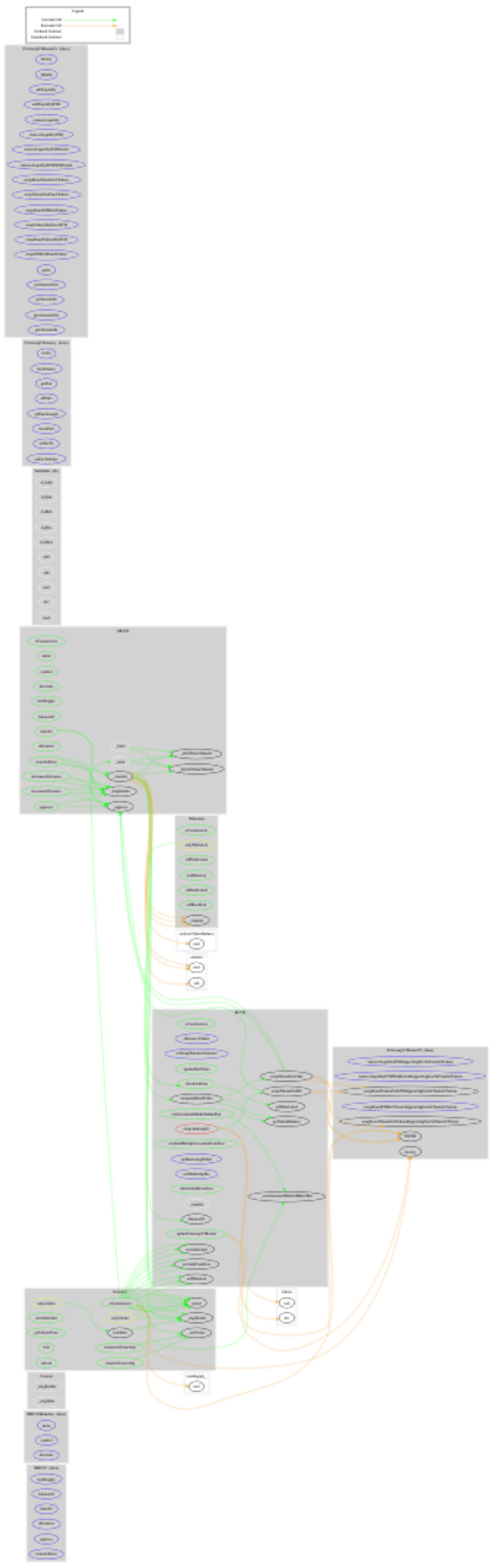
Version	Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	ECRecover	New/Create/Create2

1.0	yes					

Inheritance Graph
v1.0



CallGraph
v1.0



Scope of Work/Verify Claims

The above token Team provided us with the files that needs to be tested (Github, Bscscan, Etherscan, files, etc.). The scope of the audit is the main contract (usual the same name as team appended with .sol).

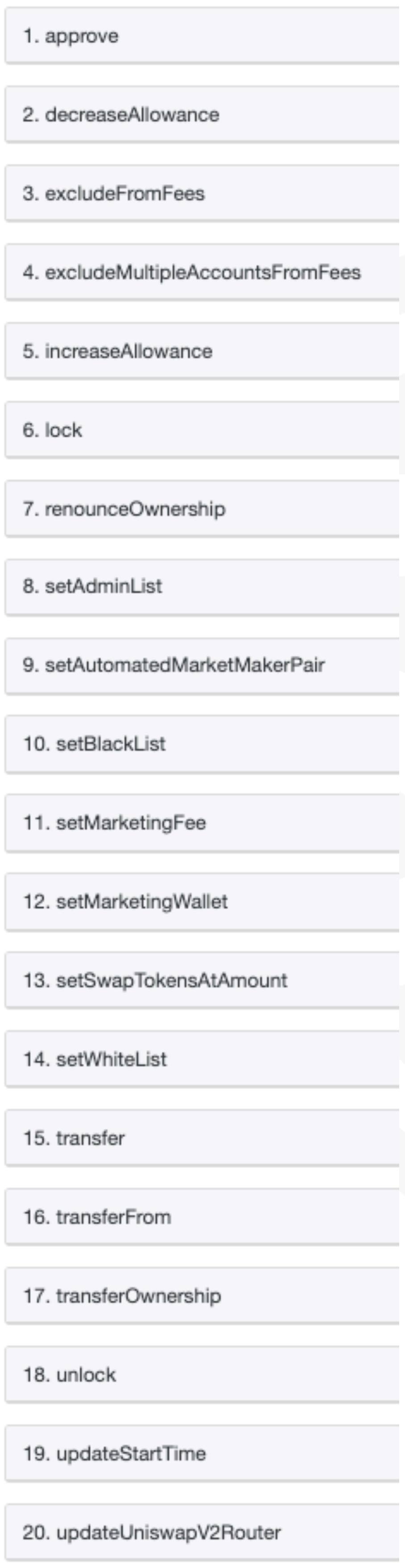
We will verify the following claims:

1. Correct implementation of Token standard
2. Deployer cannot mint any new tokens
3. Deployer cannot burn or lock user funds
4. Deployer cannot pause the contract
5. Overall checkup (Smart Contract Security)

Correct implementation of Token standard

ERC20				
Function	**Description**	**Exist**	**Tested**	**Verified**
TotalSupply	Provides information about the total token supply	✓	✓	✓
BalanceOf	Provides account balance of the owner's account	✓	✓	✓
Transfer	Executes transfers of a specified number of tokens to a specified address	✓	✓	✓
TransferFrom	Executes transfers of a specified number of tokens from a specified address	✓	✓	✓
Approve	Allow a spender to withdraw a set number of tokens from a specified account	✓	✓	✓
Allowance	Returns a set number of tokens from a spender to the owner	✓	✓	✓

Write functions of contract v1.0



1. approve
2. decreaseAllowance
3. excludeFromFees
4. excludeMultipleAccountsFromFees
5. increaseAllowance
6. lock
7. renounceOwnership
8. setAdminList
9. setAutomatedMarketMakerPair
10. setBlackList
11. setMarketingFee
12. setMarketingWallet
13. setSwapTokensAtAmount
14. setWhiteList
15. transfer
16. transferFrom
17. transferOwnership
18. unlock
19. updateStartTime
20. updateUniswapV2Router

Deployer cannot mint any new tokens

Name	Exist	Tested	Status
Deployer cannot mint	✓	✓	✓
Max / Total Supply	1.000.000.000.000.000		

Deployer cannot burn or lock user funds

Name	Exist	Tested	Status
Deployer cannot lock	✓	✓	✗
Deployer cannot burn	✓	✓	✓

Comments:

v1.0

- Deployer can lock user funds by
    - Setting start time to a high value above timestamp

Deployer cannot pause the contract

Name	Exist	Tested	Status
Deployer cannot pause	–	–	–

Overall checkup (Smart Contract Security)

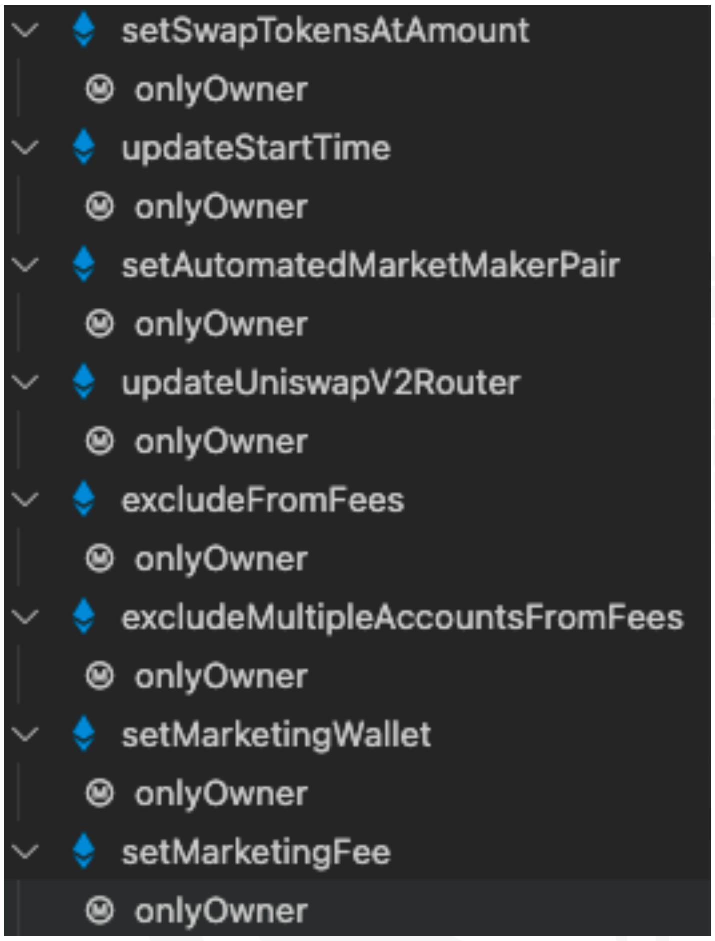
Tested	Verified
✓	✓

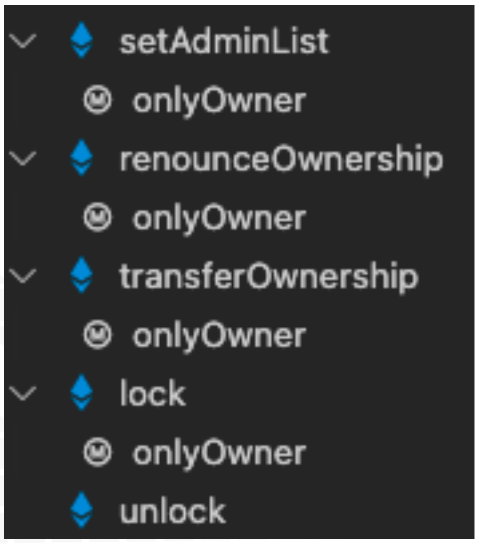
Legend

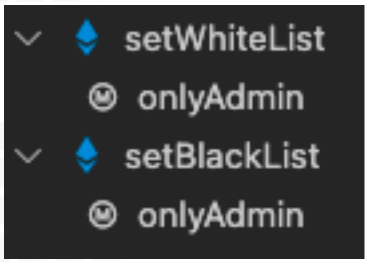
Attribute	Symbol
Verfified / Checked	✓
Partly Verified	⚑
Unverified / Not checked	✗
Not available	–

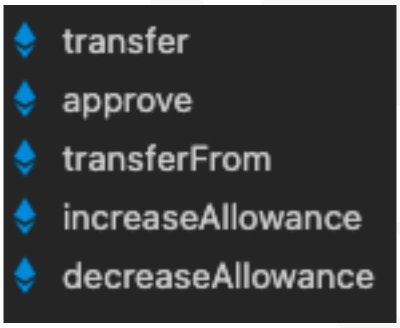
Modifiers and public functions

v1.0


setSwapTokensAtAmount
onlyOwner
updateStartTime
onlyOwner
setAutomatedMarketMakerPair
onlyOwner
updateUniswapV2Router
onlyOwner
excludeFromFees
onlyOwner
excludeMultipleAccountsFromFees
onlyOwner
setMarketingWallet
onlyOwner
setMarketingFee
onlyOwner


setAdminList
onlyOwner
renounceOwnership
onlyOwner
transferOwnership
onlyOwner
lock
onlyOwner
unlock


setWhiteList
onlyAdmin
setBlackList
onlyAdmin


transfer
approve
transferFrom
increaseAllowance
decreaseAllowance

Comments

- Deployer can set following state variables without any limitations
  - swapTokensAtAmount
  - _startTime

- Deployer can enable/disable following state variables
  - automatedMarketMakerPairs
  - _isExcludedFromFees
  - _adminList

- Deployer can set following addresses
  - uniswapV2Router

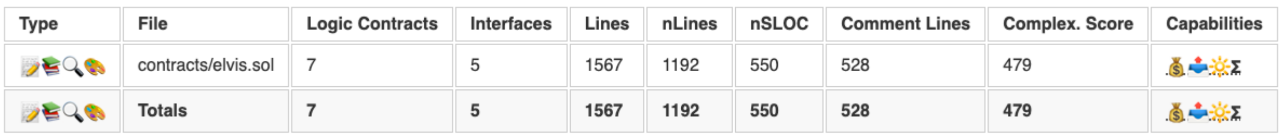
    - uniswapV2Pair
    - _marketingWalletAddress

- Deployer/admins can set
    - New whitelists
    - New blacklists

Please check if an OnlyOwner or similar restrictive modifier has been forgotten.

Source Units in Scope

v1.0



Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝📚🔍🎨	contracts/elvis.sol	7	5	1567	1192	550	528	479	💰📥🔆Σ
📝📚🔍🎨	**Totals**	**7**	**5**	**1567**	**1192**	**550**	**528**	**479**	💰📥🔆Σ

Legend

Attribute	Description
Lines	total lines of the source unit
nLines	normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
nSLOC	normalized source lines of code (only source-code lines; no comments, no blank lines)
Comment Lines	lines containing single or block comments
Complexity Score	a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

Audit Results

AUDIT PASSED

Critical issues

No critical issues

High issues

No high issues

Medium issues

Issue	File	Type	Line	Description
#1	Main	Regain ownership	See description	Owner can regain ownership after transferring it with following steps: 1. Call lock function to set _previousOwner to the own address 2. Call unlock to get ownership back 3. Transfer/renounce ownership 4. Call unlock to get ownership back Make sure to set the _previousOwnership back to address zero after using the unlock function

Low issues

Issue	File	Type	Line	Description

#1	Main	Contract doesn’t import npm packages from source (like OpenZeppelin etc.)	-	We recommend to import all packages from npm directly without flatten the contract. Functions could be modified or can be susceptible to vulnerabilities
#2	Main	A floating pragma is set	1155	The current pragma Solidity directive is „“>=0.8.4””.
#3	Main	Missing Zero Address Validation (missing-zero-check)	1361, 1327	Check that the address is not zero
#4	Main	State variable visibility is not set	1160	It is best practice to set the visibility of state variables explicitly
#5	Main	Missing Events Arithmetic	1371-1373, 1274, 1279	Emit an event for critical parameter changes
#6	Main	Locktime has no effect	604	LockTime has no effect in ownable contract because of the require statement was commented out
#7	Main	Blacklists were not used	See WhiteList contract	Blacklisted addresses were not used in the contract. Whitelisted addresses were only used in checkAddress function
#8	Main	Low level calls	1468	Check the call success with a require statement to make sure that there is no issue

Informational issues

Issue	File	Type	Line	Description
#1	Main	State variables that could be declared constant (constable-states)	1160	Add the `constant` attributes to state variables that never change
#2	Main	Functions that are not used	1474, 1497, 412	Remove unused functions
#3	Main	NatSpec documentation missing	-	If you started to comment your code, also comment all other functions, variables etc.

#4	Main	Wrong error message	1397	Error message is wrong. checkAddress is checking for whitelisted or not and not for balance overflow

Commented Code exist

There are some instances of code being commented out in the following files that should be removed:

Line	Comment
604	//require(block.timestamp > _lockTime , "Contract is locked until 7 days");
1174	//uint256 public liquidityFee;
1179	//address public liquidityWallet;
1232	//liquidityWallet = 0x44444444444444444444444444444;
1234	//liquidityFee = feeSettings[0];
1236	//totalFees =liquidityFee.add(marketingFee);

1283-1288	// function updateLiquidityWallet(address newLiquidityWallet) public onlyOwner {
	// require(newLiquidityWallet != liquidityWallet, "RedCheCoin The liquidity wallet is already this address");
	// _isExcludedFromFees[newLiquidityWallet] = true;
	// emit LiquidityWalletUpdated(newLiquidityWallet, liquidityWallet);
	// liquidityWallet = newLiquidityWallet;
	// }

1364-1368	// function setLiquiditFee(uint256 value) external onlyOwner {
	// liquidityFee = value;
	// totalFees = liquidityFee.add(marketingFee);
	// require(totalFees <= 25, "Total fee is over 25%");
	// }

1372	//totalFees = liquidityFee.add(marketingFee);

1421	// uint256 swapTokens = contractTokenBalance.mul(liquidityFee).div(
	// totalFees
	//);
	// swapAndLiquify(swapTokens);

1459-1461	//IERC20(rewardToken).balanceOf(
	//address(this)
	//);

1553-1566	// function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private { // // approve token transfer to cover all possible scenarios // _approve(address(this), address(uniswapV2Router), tokenAmount); // // add the liquidity // uniswapV2Router.addLiquidityETH{value: ethAmount}(// address(this), // tokenAmount, // 0, // slippage is unavoidable // 0, // slippage is unavoidable // liquidityWallet, // block.timestamp //); // }

Recommendation

Remove the commented code, or address them properly.

Audit Comments

We recommend you to use the special form of comments (NatSpec Format, Follow link for more information https://docs.soliditylang.org/en/v0.5.10/natspec-format.html) for your contracts to provide rich documentation for functions, return variables and more. This helps investors to make clear what that variables, functions etc. do.

15. March 2022:

- Read whole report for more information

SWC Attacks

ID	Title	Relationships	Status
SWC-136	Unencrypted Private Data On-Chain	CWE-767: Access to Critical Private Variable via Public Method	**PASSED**
SWC-135	Code With No Effects	CWE-1164: Irrelevant Code	**NOT PASSED**
SWC-134	Message call with hardcoded gas amount	CWE-655: Improper Initialization	**PASSED**
SWC-133	Hash Collisions With Multiple Variable Length Arguments	CWE-294: Authentication Bypass by Capture-replay	**PASSED**
SWC-132	Unexpected Ether balance	CWE-667: Improper Locking	**PASSED**
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	**PASSED**
SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	**PASSED**
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	**PASSED**
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	**PASSED**

SWC-127	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	**PASSED**
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	**PASSED**
SWC-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	**PASSED**
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	**PASSED**
SWC-122	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	**PASSED**
SWC-121	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	**PASSED**
SWC-120	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	**PASSED**
SWC-119	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	**PASSED**
SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization	**PASSED**
SWC-117	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	**PASSED**

SWC-116	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function	PASSED
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	PASSED
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	PASSED
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	PASSED
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	PASSED
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	PASSED
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	NOT PASSED
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	PASSED
SWC-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	PASSED

SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	PASSED
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	PASSED
SWC-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	NOT PASSED
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	PASSED
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	PASSED
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	PASSED

Solid Proofed

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY