auditace.tech

AuditAce
FROM INCEPTION TO SUCCESS

Smart Contract Audit

FOR

OHMSTRONGPresale

DATED : 8 august 23'

MANUAL TESTING

Centralization – Centralized Control Over Presale Configuration

Severity: High

function: setConfig – changeToken - ...

Status: Open

Overview:

The contract owner possesses exclusive authority to manually activate trades for investors. Without this intervention, all participants, including those with whitelisted wallets, are unable to buy, sell, or transfer their tokens. This centralized control can raise trust and security issues, particularly if the owner's intent or integrity comes into question.

```
function setConfig(...) external onlyOwner {...}
function changeToken(address _token) external onlyOwner {...}
function changeRouter(address _router) external onlyOwner {...}
function changeTreasury(address _treasury) external onlyOwner {...}
function changeTimes(uint256[2] memory _times) external onlyOwner {...}
function changePresaleRate(uint256 _rate) external onlyOwner {...}
```

Suggestion

Either set presale configurations as immutable post the initiation of the presale or devise a more decentralized approach for revising presale configurations to reduce single points of failure and enhance trustworthiness.

AUDIT SUMMARY

Project name –OHMSTRONGPresale

Date: 8 august, 2023

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

Audit Status: **Passed with High Risk**

Issues Found

Status	Critical	High	Medium	Low	Suggestion
Open	0	1	1	0	1
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.

Testnet version:
The tests were performed using the contract deployed on the BSC Testnet, which can be found at the following address:
https://testnet.bscscan.com/address/0xC2C7F9f1b1850Dd84B34286BB51599CBfAA67DF1

Token Information

Token Name : OHMSTRONGPresale

Token Symbol: -

Decimals: -

Token Supply: -

Token Address:
0x6B6dB75dd80a2820361a188eb24Eb4B4bdD9ba9f

Checksum:
3bcaa5e555bf5a31851d7f4e04fe65262ca21f14

Owner:
0x0cE88bC03b29d037DB9ACc3695D98A35E141778B
(at time of writing the audit)

Deployer:
0x6B6dB75dd80a2820361a188eb24Eb4B4bdD9ba9f

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

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

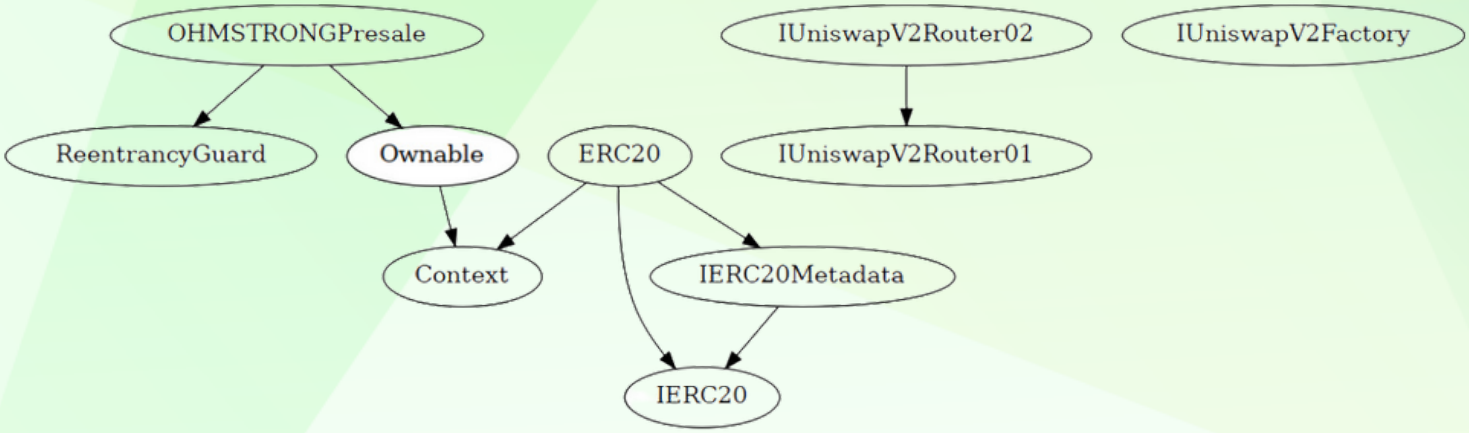
CLASSIFICATION OF RISK

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	1
Medium-Risk	1
Low-Risk	0
Gas Optimization / Suggestions	1

INHERITANCE TREE



CONTRACT ASSESMENT

Contract	Type	Bases		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
ReentrancyGuard	Implementation			
└	<Constructor>	Public ❗	🛑	NO ❗
└	_nonReentrantBefore	Private 🔐	🛑	
└	_nonReentrantAfter	Private 🔐	🛑	
└	_reentrancyGuardEntered	Internal 🔒		
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 🔒	🛑	
IERC20	Interface			
└	totalSupply	External ❗		NO ❗
└	balanceOf	External ❗		NO ❗
└	transfer	External ❗	🛑	NO ❗
└	allowance	External ❗		NO ❗

CONTRACT ASSESMENT

└	approve	External ❗	🛑	NO ❗
└	transferFrom	External ❗	🛑	NO ❗

IERC20Metadata	Interface	IERC20	
└	name	External ❗	NO ❗
└	symbol	External ❗	NO ❗
└	decimals	External ❗	NO ❗

ERC20	Implementation	Context, IERC20, IERC20Metadata	
└	<Constructor>	Public ❗	🛑 NO ❗
└	name	Public ❗	NO ❗
└	symbol	Public ❗	NO ❗
└	decimals	Public ❗	NO ❗
└	totalSupply	Public ❗	NO ❗
└	balanceOf	Public ❗	NO ❗
└	transfer	Public ❗	🛑 NO ❗
└	allowance	Public ❗	NO ❗
└	approve	Public ❗	🛑 NO ❗
└	transferFrom	Public ❗	🛑 NO ❗
└	increaseAllowance	Public ❗	🛑 NO ❗
└	decreaseAllowance	Public ❗	🛑 NO ❗
└	_transfer	Internal 🔒	🛑
└	_mint	Internal 🔒	🛑
└	_burn	Internal 🔒	🛑
└	_approve	Internal 🔒	🛑
└	_spendAllowance	Internal 🔒	🛑
└	_beforeTokenTransfer	Internal 🔒	🛑
└	_afterTokenTransfer	Internal 🔒	🛑

IUniswapV2Router01	Interface		
└	factory	External ❗	NO ❗
└	WETH	External ❗	NO ❗
└	addLiquidity	External ❗	🛑 NO ❗
└	addLiquidityETH	External ❗	💵 NO ❗
└	removeLiquidity	External ❗	🛑 NO ❗
└	removeLiquidityETH	External ❗	🛑 NO ❗
└	removeLiquidityWithPermit	External ❗	🛑 NO ❗

CONTRACT ASSESMENT

└	removeLiquidityETHWithPermit	External ❗	🛑	NO ❗
└	swapExactTokensForTokens	External ❗	🛑	NO ❗
└	swapTokensForExactTokens	External ❗	🛑	NO ❗
└	swapExactETHForTokens	External ❗	💸	NO ❗
└	swapTokensForExactETH	External ❗	🛑	NO ❗
└	swapExactTokensForETH	External ❗	🛑	NO ❗
└	swapETHForExactTokens	External ❗	💸	NO ❗
└	quote	External ❗		NO ❗
└	getAmountOut	External ❗		NO ❗
└	getAmountIn	External ❗	NO ❗
└	getAmountsOut	External ❗	NO ❗
└	getAmountsIn	External ❗		NO ❗
IUniswapV2Router02	Interface	IUniswapV2Router01		
└	removeLiquidityETHSupportingFeeOnTransferTokens	External ❗	🛑	NO ❗
└	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External ❗	🛑	NO ❗
└	swapExactTokensForTokensSupportingFeeOnTransferTokens	External ❗	🛑	NO ❗
└	swapExactETHForTokensSupportingFeeOnTransferTokens	External ❗	💸	NO ❗
└	swapExactTokensForETHSupportingFeeOnTransferTokens	External ❗	🛑	NO ❗
IUniswapV2Factory	Interface			
└	feeTo	External ❗		NO ❗
└	feeToSetter	External ❗		NO ❗
└	getPair	External ❗	NO ❗
└	allPairs	External ❗	NO ❗
└	allPairsLength	External ❗	NO ❗
└	createPair	External ❗	🛑	NO ❗
└	setFeeTo	External ❗	🛑	NO ❗
└	setFeeToSetter	External ❗	🛑	NO ❗

CONTRACT ASSESMENT

OHMSTRONGPresale	Implementation	Ownable, ReentrancyGuard		
└	<Constructor>	Public ❗	🛑	NO ❗
└	setConfig	External ❗	🛑	onlyOwner
└	changeToken	External ❗	🛑	onlyOwner
└	changeRouter	External ❗	🛑	onlyOwner
└	changeTreasury	External ❗	🛑	onlyOwner
└	changeTimes	External ❗	🛑	onlyOwner
└	changePresaleRate	External ❗	🛑	onlyOwner
└	invest	External ❗	💵	nonReentrant
└	claimToken	External ❗	🛑	nonReentrant
└	finishSale	External ❗	🛑	onlyOwner nonReentrant
└	getContractInfo	External ❗		NO ❗
└	getPresaleSuccess	External ❗		NO ❗
└	addLiquidity	Internal 🔒	🛑	
└	withdrawToken	External ❗	🛑	onlyOwner
└	_isSatisfySale	Private 🔐		

Legend

Symbol	Meaning
🛑	Function can modify state
💵	Function is payable

STATIC ANALYSIS

```
INFO:Detectors:
Pragma version^0.8.0 (contracts/Token.sol#8) allows old versions
Pragma version^0.8.0 (contracts/Token.sol#87) allows old versions
Pragma version^0.8.0 (contracts/Token.sol#113) allows old versions
Pragma version^0.8.0 (contracts/Token.sol#202) allows old versions
Pragma version^0.8.0 (contracts/Token.sol#293) allows old versions
Pragma version^0.8.0 (contracts/Token.sol#321) allows old versions
Pragma version^0.8.17 (contracts/Token.sol#732) allows old versions
solc-0.8.17 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Function IUniswapV2Router01.WETH() (contracts/Token.sol#737) is not in mixedCase
Event OHMSTRONGPresale.userDeposit(uint256,address) (contracts/Token.sol#985) is not in CapWords
Event OHMSTRONGPresale.userRefunded(uint256,address) (contracts/Token.sol#986) is not in CapWords
Event OHMSTRONGPresale.userClaimed(uint256,address) (contracts/Token.sol#987) is not in CapWords
Parameter OHMSTRONGPresale.setConfig(uint256[2],uint256[2],uint256[2],uint256,uint256[2],address[2],address)._caps (contracts/Token.sol#995) is not in mixedCase
Parameter OHMSTRONGPresale.setConfig(uint256[2],uint256[2],uint256[2],uint256,uint256[2],address[2],address)._times (contracts/Token.sol#996) is not in mixedCase
Parameter OHMSTRONGPresale.setConfig(uint256[2],uint256[2],uint256[2],uint256,uint256[2],address[2],address)._rates (contracts/Token.sol#997) is not in mixedCase
Parameter OHMSTRONGPresale.setConfig(uint256[2],uint256[2],uint256[2],uint256,uint256[2],address[2],address)._lpPercent (contracts/Token.sol#998) is not in mixedCase
Parameter OHMSTRONGPresale.setConfig(uint256[2],uint256[2],uint256[2],uint256,uint256[2],address[2],address)._limits (contracts/Token.sol#999) is not in mixedCase
Parameter OHMSTRONGPresale.setConfig(uint256[2],uint256[2],uint256[2],uint256,uint256[2],address[2],address)._adds (contracts/Token.sol#1000) is not in mixedCase
Parameter OHMSTRONGPresale.setConfig(uint256[2],uint256[2],uint256[2],uint256,uint256[2],address[2],address)._ohmstrong (contracts/Token.sol#1001) is not in mixedCase
Parameter OHMSTRONGPresale.changeToken(address)._token (contracts/Token.sol#1017) is not in mixedCase
Parameter OHMSTRONGPresale.changeRouter(address)._router (contracts/Token.sol#1021) is not in mixedCase
Parameter OHMSTRONGPresale.changeTreasury(address)._treasury (contracts/Token.sol#1025) is not in mixedCase
Parameter OHMSTRONGPresale.changeTimes(uint256[2])._times (contracts/Token.sol#1029) is not in mixedCase
Parameter OHMSTRONGPresale.changePresaleRate(uint256)._rate (contracts/Token.sol#1034) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Reentrancy in OHMSTRONGPresale.finishSale() (contracts/Token.sol#1075-1109):
        External calls:
        - address(devWallet).transfer(20000000000000000) (contracts/Token.sol#1096)
        - address(devWallet).transfer(balance) (contracts/Token.sol#1098)
        - address(teamWallet).transfer(balance) (contracts/Token.sol#1101)
        External calls sending eth:
        - addLiquidity(ethAmountForLiquidity,tokenAmountforliquidity) (contracts/Token.sol#1084)
                - (None,None,liquidity) = IUniswapV2Router02(router).addLiquidityETH{value: ethAmount}(ohmstrong,tokenAmount,0,0,address(this),block.timestamp + 60) (contracts/Token.sol#1149-1151)
        - address(devWallet).transfer(20000000000000000) (contracts/Token.sol#1096)
        - address(devWallet).transfer(balance) (contracts/Token.sol#1098)
        - address(teamWallet).transfer(balance) (contracts/Token.sol#1101)
        State variables written after the call(s):
        - isFinalized = true (contracts/Token.sol#1108)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4
INFO:Detectors:
Variable IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountADesired (contracts/Token.sol#742) is too similar to IUniswapV2Router01.addLiquidity(addr
ess,address,uint256,uint256,uint256,uint256,address,uint256).amountBDesired (contracts/Token.sol#743)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar
INFO:Detectors:
OHMSTRONGPresale.devWallet (contracts/Token.sol#973) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
```

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

No major issues were found in the output

FUNCTIONAL TESTING

1- Setting config (passed):

https://testnet.bscscan.com/tx/0x66af8ecdebdf92ff15be738569f77184bf2da5fbe3aee6e8eb945134394cd8e8

2- Investing (passed):

- Was enabled after start time
- Account states (total earned and total deposited) updated correctly
- Contract state (total deposited ETH) updated correctly
- Invested using multiple accounts
- Received correct amount of tokens (568.15 tokens per ETH, invested 0.1ETH and received 56.815 tokens)

https://testnet.bscscan.com/tx/0xc5bdfe726477fdadc3025efa00a8a4018f191b2392f58a6c8a86c0bbab2b5905

3- Finishing Sale (passed):

- Finished presale after reaching hardcap
- Liquidity added successfully
- Remaining tokens (excluding total purchased) were sent to owner

https://testnet.bscscan.com/tx/0x4a0bfe5adfbc1c554743cd531e233ff59d619fd52b4062f6e3839d4b8ebb1f69

4- Claiming Tokens (passed):

- Received right amount of tokens
- Couldn't claim excessive tokens
- Only could claim after end of presale (finalized)

https://testnet.bscscan.com/tx/0x18af936a3ffd555c8c3fd80ea831d7adfc46e5852a098987e62412f0678d1afc

MANUAL TESTING

Centralization – Centralized Control Over Presale Configuration

Severity: High

function: setConfig – changeToken - ...

Status: Open

Overview:

The contract owner possesses exclusive authority to manually activate trades for investors. Without this intervention, all participants, including those with whitelisted wallets, are unable to buy, sell, or transfer their tokens. This centralized control can raise trust and security issues, particularly if the owner's intent or integrity comes into question.

```
function setConfig(...) external onlyOwner {...}
function changeToken(address _token) external onlyOwner {...}
function changeRouter(address _router) external onlyOwner {...}
function changeTreasury(address _treasury) external onlyOwner {...}
function changeTimes(uint256[2] memory _times) external onlyOwner {...}
function changePresaleRate(uint256 _rate) external onlyOwner {...}
```

Suggestion

Either set presale configurations as immutable post the initiation of the presale or devise a more decentralized approach for revising presale configurations to reduce single points of failure and enhance trustworthiness.

MANUAL TESTING

Logical — Unauthorized Token Claims Post-Presale End

Severity: Medium

function: invest

Status: Open

Overview:

If the presale concludes before the designated end time, a loophole allows users to claim tokens from the staking contract. This vulnerability poses a substantial risk, especially when the presale address holds a vast amount of tokens. Exploiters can leverage this to obtain tokens at significantly discounted prices.

Suggestion

Implement a mechanism to prevent users from depositing tokens once the presale is finalized.

MANUAL TESTING

Logical – Potential flash-loan attack

Severity: Informational (For discussion)

function: finishSale

Status: Open

Overview:

The `finishSale` function, as per the latest pool configuration, contributes 50% of the amassed ETH to the Uniswap v2 ETH/ohmstrong pool. A vulnerability arises when attackers monitor the txpool for the `finishSale` transaction. Once the function is called, they can leverage flash loans to acquire a large amount of ETH, purchase any remaining tokens from the presale contract, and subsequently sell them on Uniswap v2 for a more substantial ETH sum. The extent of this vulnerability's impact is contingent on the ohmstrong token contract's implementation and the token's initial price, making its repercussions uncertain.

However, it's worth noting that the system limits a single address's token purchase to the `maxBuyPerParticipant` constraint, currently set at 1 ETH. While this substantially reduces the potential attack vector, sophisticated contract systems can still facilitate the exploit.

Suggestion

Institute a waiting period post the `finishSale` function call during which participants cannot claim or invest in tokens.

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