auditace.tech

AuditAce
FROM INCEPTION TO SUCCESS

Smart Contract Audit

FOR

Starship Doge

DATED : 25 Apr 23'

AUDIT SUMMARY

Project name – Starship Doge

Date: 25 April, 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	0	0	0
Acknowledged	0	0	0	0	0
Resolved	0	0	0	0	0

USED TOOLS

Tools:

1.Manual Review: The code has undergone a line-by-line review by the Ace team.

2.Slither: The code has undergone static analysis using Slither.

Note:

It is important to note that any issues or vulnerabilities identified during the audit are not the responsibility of the auditor, as the **STARSHIPDOGE** is already live and actively traded. The auditor's role is limited to providing an independent evaluation of the smart contract code, as provided by the token's development team, and identifying potential issues or areas for improvement.

Token Information

Token Name : Starship Doge

Token Symbol: STARSHIPDOGE

Decimals: 9

Token Supply: 1,000,000,000,000,000

Token Address:
0x67e4CFD91E4bAa4A6794c7974FeC892247B26D3C

Checksum:
a57198f035f197a287e782cfefdb4f2dc22dd4ed

Owner:
0x0000000000000000000000000000000000000000
(RENOUNCED)

TOKEN OVERVIEW

Fees:

Buy Fees: 10%

Sell Fees: 10%

Transfer Fees: 10%

Fees Privilige: none

Ownership : renounced

Minting: No mint function

Max Tx Amount/ Max Wallet Amount: No

Blacklist: no

Other Priviliges: none

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

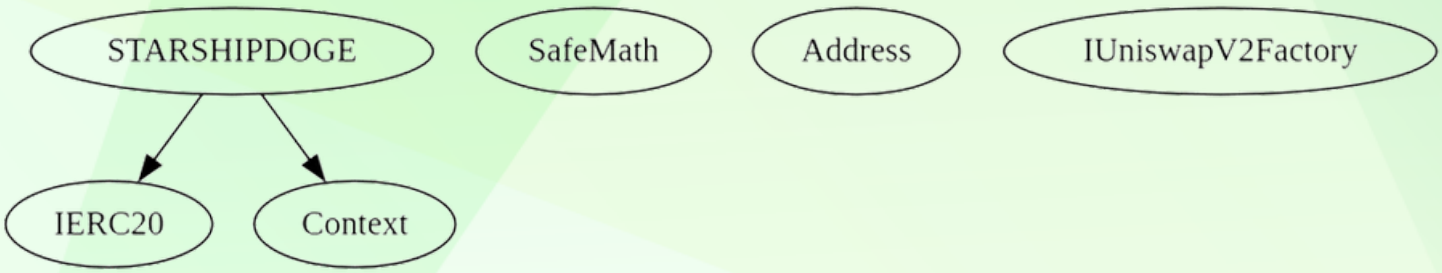
CLASSIFICATION OF RISK

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	0
Low-Risk	0
Gas Optimization / Suggestions	0

INHERITANCE TREE



POINTS TO NOTE

- Ownership is renounced, meaning owner has not control over the contract functions
- Owner is not able to modify buy/sell/transfer fees (10% for each)
- Owner is not able to set max buy/sell/transfer/hold amount
- Owner is not able to blacklist an arbitrary wallet
- Owner is not able to disable trades
- Owner is not able to mint new tokens

CONTRACT ASSESMENT

Contract	Type	Bases		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
└	totalSupply	External ❗		NO❗
└	balanceOf	External ❗		NO❗
└	transfer	External ❗	🛑	NO❗
└	allowance	External ❗		NO❗
└	approve	External ❗	🛑	NO❗
└	transferFrom	External ❗	🛑	NO❗
SafeMath	Library			
└	add	Internal 🔒		
└	sub	Internal 🔒		
└	mul	Internal 🔒		
└	div	Internal 🔒		
└	sub	Internal 🔒		
└	div	Internal 🔒		
Context	Implementation			
└	_msgSender	Internal 🔒		
└	_msgData	Internal 🔒		
Address	Library			
└	isContract	Internal 🔒		
└	sendValue	Internal 🔒	🛑	
└	functionCall	Internal 🔒	🛑	
└	functionCall	Internal 🔒	🛑	
└	functionCallWithValue	Internal 🔒	🛑	
└	functionCallWithValue	Internal 🔒	🛑	
└	functionStaticCall	Internal 🔒		
└	functionStaticCall	Internal 🔒		
└	functionDelegateCall	Internal 🔒	🛑	
└	functionDelegateCall	Internal 🔒	🛑	
└	_verifyCallResult	Private 🔐		
IUniswapV2Factory	Interface			
└	feeTo	External ❗		NO❗
└	feeToSetter	External ❗		NO❗
└	getPair	External ❗		NO❗
└	allPairs	External ❗		NO❗
└	allPairsLength	External ❗		NO❗

CONTRACT ASSESMENT

└	createPair	External ❗	🛑	NO❗
└	setFeeTo	External ❗	🛑	NO❗
└	setFeeToSetter	External ❗	🛑	NO❗
IUniswapV2Pair	Interface				
└	name	External ❗		NO❗
└	symbol	External ❗		NO❗
└	decimals	External ❗		NO❗
└	totalSupply	External ❗		NO❗
└	balanceOf	External ❗		NO❗
└	allowance	External ❗		NO❗
└	approve	External ❗	🛑	NO❗
└	transfer	External ❗	🛑	NO❗
└	transferFrom	External ❗	🛑	NO❗
└	DOMAIN_SEPARATOR	External ❗		NO❗
└	PERMIT_TYPEHASH	External ❗		NO❗
└	nonces	External ❗		NO❗
└	permit	External ❗	🛑	NO❗
└	MINIMUM_LIQUIDITY	External ❗		NO❗
└	factory	External ❗		NO❗
└	token0	External ❗		NO❗
└	token1	External ❗		NO❗
└	getReserves	External ❗		NO❗
└	price0CumulativeLast	External ❗		NO❗
└	price1CumulativeLast	External ❗		NO❗
└	kLast	External ❗		NO❗
└	burn	External ❗	🛑	NO❗
└	swap	External ❗	🛑	NO❗
└	skim	External ❗	🛑	NO❗
└	sync	External ❗	🛑	NO❗
└	initialize	External ❗	🛑	NO❗
IUniswapV2Router01	Interface				
└	factory	External ❗		NO❗
└	WETH	External ❗		NO❗
└	addLiquidity	External ❗	🛑	NO❗
└	addLiquidityETH	External ❗	💵	NO❗
└	removeLiquidity	External ❗	🛑	NO❗
└	removeLiquidityETH	External ❗	🛑	NO❗
└	removeLiquidityWithPermit	External ❗	🛑	NO❗
└	removeLiquidityETHWithPermit	External ❗	🛑	NO❗

CONTRACT ASSESMENT

└	swapExactTokensForTokens	External ❗	🛑	NO❗
└	swapTokensForExactTokens	External ❗	🛑	NO❗
└	swapExactETHForTokens	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❗
STARSHIPDOGE	Implementation	Context, IERC20		
└	owner	Public ❗		NO❗
└	renounceOwnership	Public ❗	🛑	NO❗
└	<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❗
└	<Receive Ether>	External ❗	💵	NO❗
└	_getCurrentSupply	Private 🔐		
└	_approve	Private 🔐	🛑	
└	_transfer	Private 🔐	🛑	
└	sendToWallet	Private 🔐	🛑	
└	swapAndLiquify	Private 🔐	🛑	lockTheSwap
└	swapTokensForBNB	Private 🔐	🛑	
└	addLiquidity	Private 🔐	🛑	
└	remove_Random_Tokens	Public ❗	🛑	NO❗
└	_tokenTransfer	Private 🔐	🛑	

STATIC ANALYSIS

```
Reentrancy in STARSHIPDOGE.transferFrom(address,address,uint256) (contracts/Token.sol#667-682):
        External calls:
        - _transfer(sender,recipient,amount) (contracts/Token.sol#672)
                - wallet.transfer(amount) (contracts/Token.sol#778)
        External calls sending eth:
        - _transfer(sender,recipient,amount) (contracts/Token.sol#672)
                - wallet.transfer(amount) (contracts/Token.sol#778)
                - uniswapV2Router.addLiquidityETH{value: BNBAmount}(address(this),tokenAmount,0,0,Wallet_Burn,block.timestamp) (contracts/Token.sol#826-833)
        State variables written after the call(s):
        - _approve(sender,_msgSender(),_allowances[sender][_msgSender()].sub(amount,ERC20: transfer amount exceeds allowance)) (contracts/Token.sol#673-680)
                - _allowances[theOwner][theSpender] = amount (contracts/Token.sol#726)
        Event emitted after the call(s):
        - Approval(theOwner,theSpender,amount) (contracts/Token.sol#727)
                - _approve(sender,_msgSender(),_allowances[sender][_msgSender()].sub(amount,ERC20: transfer amount exceeds allowance)) (contracts/Token.sol#673-680)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4

Variable IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountADesired (contracts/Token.sol#353) is too similar to IUniswapV2Router01.addLiquidit
y(address,address,uint256,uint256,uint256,uint256,address,uint256).amountBDesired (contracts/Token.sol#354)
Variable STARSHIPDOGE.swapAndLiquify(uint256).tokens_to_D (contracts/Token.sol#787) is too similar to STARSHIPDOGE.swapAndLiquify(uint256).tokens_to_M (contracts/Token.sol#786)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar

STARSHIPDOGE.slitherConstructorVariables() (contracts/Token.sol#541-881) uses literals with too many digits:
        - _tTotal = 10000000000000 * 10 ** 6 * 10 ** 6 * 10 ** _decimals (contracts/Token.sol#575-576)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

STARSHIPDOGE.MAX (contracts/Token.sol#573) is never used in STARSHIPDOGE (contracts/Token.sol#541-881)
STARSHIPDOGE._previousMaxWalletToken (contracts/Token.sol#588) is never used in STARSHIPDOGE (contracts/Token.sol#541-881)
STARSHIPDOGE._previousMaxTxAmount (contracts/Token.sol#590) is never used in STARSHIPDOGE (contracts/Token.sol#541-881)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable

STARSHIPDOGE.Percent_AutoLP (contracts/Token.sol#586) should be constant
STARSHIPDOGE.Percent_Burn (contracts/Token.sol#585) should be constant
STARSHIPDOGE.Percent_Dev (contracts/Token.sol#584) should be constant
STARSHIPDOGE.Percent_Marketing (contracts/Token.sol#583) should be constant
STARSHIPDOGE.Wallet_Dev (contracts/Token.sol#569-570) should be constant
STARSHIPDOGE.Wallet_Marketing (contracts/Token.sol#567-568) should be constant
STARSHIPDOGE._Tax_On_Buy (contracts/Token.sol#581) should be constant
STARSHIPDOGE._Tax_On_Sell (contracts/Token.sol#582) should be constant
STARSHIPDOGE.swapAndLiquifyEnabled (contracts/Token.sol#594) should be constant
STARSHIPDOGE.swapTrigger (contracts/Token.sol#580) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant

STARSHIPDOGE._maxTxAmount (contracts/Token.sol#589) should be immutable
STARSHIPDOGE._maxWalletToken (contracts/Token.sol#587) should be immutable
STARSHIPDOGE._previousMaxTxAmount (contracts/Token.sol#590) should be immutable
STARSHIPDOGE._previousMaxWalletToken (contracts/Token.sol#588) should be immutable
STARSHIPDOGE.uniswapV2Pair (contracts/Token.sol#592) should be immutable
STARSHIPDOGE.uniswapV2Router (contracts/Token.sol#591) 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

As the token is already live and actively traded, with no apparent issues affecting its functionality or trading, we have determined that performing unit tests on the smart contract is not necessary for this audit. Our focus has been on reviewing the smart contract code and its compliance with the ERC20 standard, as well as identifying potential vulnerabilities and areas for improvement.

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