AuditAce
FROM INCEPTION TO SUCCESS

Smart Contract Audit

FOR

BassDoge

DATED : 26 June 24'

MANUAL TESTING

Centralization – Marketing and Liquidity Fees.

Severity: High

Function: setMarketingFee, setLiquidityFee

Status: FIXED

Overview:

The owner can set the buy and sell fees up to 100%, which is not recommended.

```
function setLiquidityFee(uint256 value) external onlyOwner {
    liquidityFee = value;
    totalFees = marketingFee.add(liquidityFee);
  }
function setMarketingFee(uint256 value) external onlyOwner {
    marketingFee = value;
    totalFees = marketingFee.add(liquidityFee);
  }
```

MANUAL TESTING

Centralization – Missing Require Check.

Severity: High

Function: setMarketingWallet

Status: FIXED

Overview:

The owner can set any arbitrary address excluding zero address as this is not recommended because if the owner sets the address to the contract address, then the ETH will not be sent to that address and the transaction will fail and this will lead to a potential honeypot in the contract.

```
function setMarketingWallet(address payable wallet) external onlyOwner {
    marketingWalletAddress = wallet;
}
```

Suggestion:

It is recommended that the address should not be able to be set as a contract address.

AUDIT SUMMARY

Project name – BassDoge

Date: 26 June, 2024

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	2	3
Acknowledged	0	0	0	0	0
Resolved	0	2	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/0x36b6837a83c676c631cb1d05da4132d3bdd4dcc7#code

Token Information

Token Address:
0x72c1D2DF4C420a7335c1BD092110B8F8874C9997

Name: Bass Doge

Symbol: BASSDOGE

Decimals: 18

Network: BscScan

Token Type: BEP-20

Owner: 0x0000000000000000000000000000000000000000

Deployer:
0xF835075DC09ae4b4211F41bc720Ad89d5b29FeAC

Token Supply: 10000000000

Checksum: A17acbefe2a12642d388659dffd20231

Testnet:
https://testnet.bscscan.com/address/0x36b6837a83c676c631cb1d05da4132d3bdd4dcc7#code

TOKEN OVERVIEW

Liquidity Fee: 3%

Marketing Fee: 2%

Transfer Fee: 0%

Fee Privilege: Owner

Ownership: Renounced

Minting: None

Max Tx: No

Blacklist: No

AUDIT METHODOLOGY

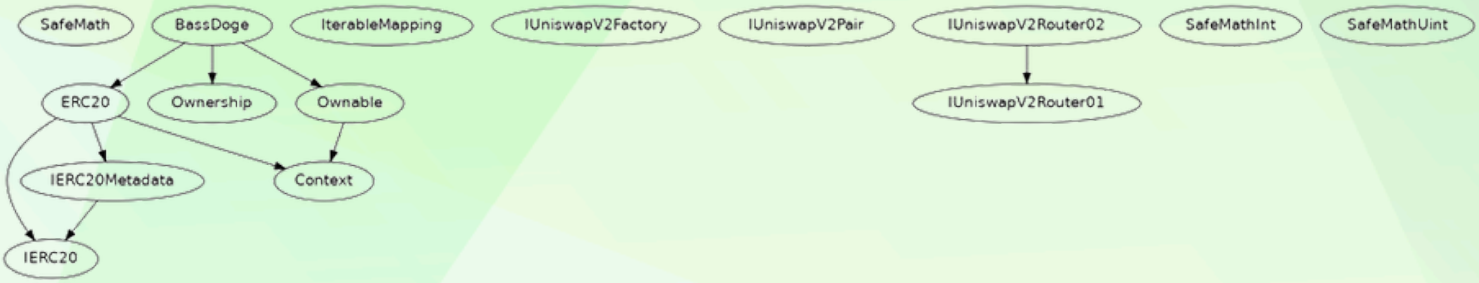
The auditing process will follow a routine as special considerations by Auditace:

- 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

INHERITANCE TREE



POINTS TO NOTE

- The owner can transfer ownership.
- The owner can renounce ownership.
- The owner can update uniswap router address.
- The owner can exclude the address from fees.
- The owner can set a marketing wallet address.
- The owner set Liquidity/Marketing fees more than 100%.
- The owner can collect all taxes.

STATIC ANALYSIS

```
INFO:Detectors:
BassDoge.constructor(string,string,uint256,uint8,uint256,address,uint256,uint256,address,address,BassDoge.RefInfo) (BassDoge.sol#1310-1356) performs a multi
plication on the result of a division:
        - swapTokensAtAmount = supply_.div(100000) * (10 ** decimals_) + 100 * (10 ** decimals_) (BassDoge.sol#1343-1347)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
BassDoge.addLiquidity(uint256,uint256) (BassDoge.sol#1547-1560) ignores return value by uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),toke
nAmount,0,0,address(0),block.timestamp) (BassDoge.sol#1552-1559)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
BassDoge.setLiquidityFee(uint256) (BassDoge.sol#1394-1397) should emit an event for:
        - totalFees = marketingFee.add(liquidityFee) (BassDoge.sol#1396)
BassDoge.setMarketingFee(uint256) (BassDoge.sol#1399-1402) should emit an event for:
        - marketingFee = value (BassDoge.sol#1400)
        - totalFees = marketingFee.add(liquidityFee) (BassDoge.sol#1401)
BassDoge.setSwapTokensAtAmount(uint256) (BassDoge.sol#1404-1406) should emit an event for:
        - swapTokensAtAmount = value (BassDoge.sol#1405)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic
INFO:Detectors:
Ownable.constructor().msgSender (BassDoge.sol#1107) lacks a zero-check on :
                - _owner = msgSender (BassDoge.sol#1108)
BassDoge.constructor(string,string,uint256,uint8,uint256,address,uint256,uint256,address,address,BassDoge.RefInfo).addr_ (BassDoge.sol#1320) lacks a zero-ch
eck on :
                - address(addr_).transfer(msg.value - ref_amount) (BassDoge.sol#1324)
BassDoge.constructor(string,string,uint256,uint8,uint256,address,uint256,uint256,address,address,BassDoge.RefInfo).marketingWalletAddress_ (BassDoge.sol#131
6) lacks a zero-check on :
                - marketingWalletAddress = marketingWalletAddress_ (BassDoge.sol#1326)
BassDoge.updateUniswapV2Router(address)._uniswapV2Pair (BassDoge.sol#1368-1369) lacks a zero-check on :
                - uniswapV2Pair = _uniswapV2Pair (BassDoge.sol#1370)
BassDoge.setMarketingWallet(address).wallet (BassDoge.sol#1390) lacks a zero-check on :
                - marketingWalletAddress = wallet (BassDoge.sol#1391)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in BassDoge._transfer(address,address,uint256) (BassDoge.sol#1434-1494):
        External calls:
        - swapAndSendToFee(marketingTokens,marketingWalletAddress) (BassDoge.sol#1467)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (BassDoge.sol#1538-15
44)
        - swapAndLiquify(swapTokens) (BassDoge.sol#1473)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,address(0),block.timestamp) (BassDoge.sol#1552-1559)
```

```
INFO:Detectors:
Variable IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountADesired (BassDoge.sol#877) is too similar t
o IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountBDesired (BassDoge.sol#878)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar
INFO:Detectors:
BassDoge.constructor(string,string,uint256,uint8,uint256,address,uint256,uint256,address,address,BassDoge.RefInfo) (BassDoge.sol#1310-1356) uses literals wi
th too many digits:
        - swapTokensAtAmount = supply_.div(100000) * (10 ** decimals_) + 100 * (10 ** decimals_) (BassDoge.sol#1343-1347)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
SafeMathInt.MAX_INT256 (BassDoge.sol#1184) is never used in SafeMathInt (BassDoge.sol#1182-1239)
BassDoge._isExcludedFromMaxTx (BassDoge.sol#1278) is never used in BassDoge (BassDoge.sol#1253-1572)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable
INFO:Detectors:
BassDoge.Optimization (BassDoge.sol#1254) should be constant
BassDoge.extraSellFee (BassDoge.sol#1268) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
BassDoge.maxTxAmount (BassDoge.sol#1272) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
INFO:Slither:BassDoge.sol analyzed (15 contracts with 93 detectors), 47 result(s) found
```

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- Approve (passed):

https://testnet.bscscan.com/tx/0xaa028f166c1197e00f56fb6fb12ea6e50af10ab035962b56c1fdfb6946cab439

2- Se Liquidity Fee (passed):

https://testnet.bscscan.com/tx/0xaf1b969758ceecf88bde91fbc4847469fcd0e42b4f9c42bfd41bba2ce300f2eb

3- Set Marketing Fee (passed):

https://testnet.bscscan.com/tx/0xf38e647bc625d8390473490172 27b9dc90d765d000dc0849f9b774116efe3932

4- Set Marketing Wallet (passed):

https://testnet.bscscan.com/tx/0xf55b1d43684565daf7fc10813f3373b109899012bd0a14d74eaede48ea0488a4

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	2
Medium-Risk	0
Low-Risk	2
Gas Optimization / Suggestions	3

MANUAL TESTING

Centralization – Marketing and Liquidity Fees.

Severity: High

Function: setMarketingFee, setLiquidityFee

Status: FIXED

Overview:

The owner can set the buy and sell fees up to 100%, which is not recommended.

```
function setLiquidityFee(uint256 value) external onlyOwner {
    liquidityFee = value;
    totalFees = marketingFee.add(liquidityFee);
  }
function setMarketingFee(uint256 value) external onlyOwner {
    marketingFee = value;
    totalFees = marketingFee.add(liquidityFee);
  }
```

MANUAL TESTING

Centralization – Missing Require Check.

Severity: High

Function: setMarketingWallet

Status: FIXED

Overview:

The owner can set any arbitrary address excluding zero address as this is not recommended because if the owner sets the address to the contract address, then the ETH will not be sent to that address and the transaction will fail and this will lead to a potential honeypot in the contract.

```
function setMarketingWallet(address payable wallet) external onlyOwner {
    marketingWalletAddress = wallet;
}
```

Suggestion:

It is recommended that the address should not be able to be set as a contract address.

MANUAL TESTING

Centralization – Missing Events

Severity: Low

Subject: Missing Events

Status: Open

Overview:
They serve as a mechanism for emitting and recording data onto the blockchain, making it transparent and easily accessible.

```
function setMarketingWallet(address payable wallet) external onlyOwner {
    marketingWalletAddress = wallet;
  }
function setMarketingFee(uint256 value) external onlyOwner {
    marketingFee = value;
    totalFees = marketingFee.add(liquidityFee);
  }
function setLiquidityFee(uint256 value) external onlyOwner {
    liquidityFee = value;
    totalFees = marketingFee.add(liquidityFee);
  }
```

Suggestion:
Emit an event for critical changes.

MANUAL TESTING

Centralization – Missing Zero Address

Severity: Low

Subject: Zero Check

Status: Open

Overview:

Functions can take a zero address as a parameter (0x00000...). If a function parameter of address type is not properly validated by checking for zero addresses, there could be serious consequences for the contract's functionality.

```
function setMarketingWallet(address payable wallet) external onlyOwner {
    marketingWalletAddress = wallet;
  }
function updateUniswapV2Router(address newAddress) public onlyOwner {
    require(
      newAddress != address(uniswapV2Router),
      "The router already has that address"
    );
    emit UpdateUniswapV2Router(newAddress, address(uniswapV2Router));
    uniswapV2Router = IUniswapV2Router02(newAddress);
    address _uniswapV2Pair = IUniswapV2Factory(uniswapV2Router.factory())
      .createPair(address(this), uniswapV2Router.WETH());
    uniswapV2Pair = _uniswapV2Pair;
  }
```

MANUAL TESTING

Optimization

Severity: Informational

Subject: Floating Pragma Solidity version

Status: Open

Overview:

It is considered best practice to pick one compiler version and stick with it. With a floating pragma, contracts may accidentally be deployed using an outdated.

pragma solidity ^0.8.18;

Suggestion:

Adding the latest constant version of solidity is recommended, as this prevents the unintentional deployment of a contract with an outdated compiler that contains unresolved bugs.

MANUAL TESTING

Optimization

Severity: Informational

Subject: Remove Safe Math

Status: Open

Line: 12-165

Overview:

compiler version above 0.8.0 can control arithmetic overflow/underflow, it is recommended to remove the unwanted code to avoid high gas fees.

MANUAL TESTING

Optimization

Severity: Optimization

Subject: Remove unused code.

Status: Open

Overview:

Unused variables are allowed in Solidity, and they do. not pose a direct security issue. It is the best practice. though to avoid them.

```
function _msgData() internal view virtual returns (bytes memory) {
    this; // silence state mutability warning without generating bytecode - see
https://github.com/ethereum/solidity/issues/2691
    return msg.data;
  }
}
function _burn(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: burn from the zero address");

    _beforeTokenTransfer(account, address(0), amount);

    _balances[account] = _balances[account].sub(
      amount,
      "ERC20: burn amount exceeds balance"
    );
    _totalSupply = _totalSupply.sub(amount);
    emit Transfer(account, address(0), amount);
  }
```

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