



Cyberscope

Audit Report

Dogebank

March 2023

Network BSC

Address 0x796A6dc7e14e09bF82b4B68bE54E9c3E3f231C3d

Audited by © cyberscope

Table of Contents

Table of Contents	1
Review	3
Audit Updates	3
Source Files	3
Findings Breakdown	4
Analysis	5
Diagnostics	6
RPI - Redundant Pair Initializations	7
Description	7
Recommendation	7
RC - Redundant Calculations	8
Description	8
Recommendation	9
PTAI - Potential Transfer Amount Inconsistency	10
Description	10
Recommendation	11
GO - Gas Optimization	12
Description	12
Recommendation	13
PAV - Phishing Attack Vulnerability	14
Description	14
Recommendation	14
IDI - Immutable Declaration Improvement	15
Description	15
Recommendation	15
L02 - State Variables could be Declared Constant	16
Description	16
Recommendation	16
L04 - Conformance to Solidity Naming Conventions	17
Description	17
Recommendation	18
L09 - Dead Code Elimination	19
Description	19
Recommendation	20
L16 - Validate Variable Setters	21
Description	21
Recommendation	21
L17 - Usage of Solidity Assembly	22
Description	22

Recommendation	22
L20 - Succeeded Transfer Check	23
Description	23
Recommendation	23
Functions Analysis	24
Inheritance Graph	31
Flow Graph	32
Summary	33
Disclaimer	34
About Cyberscope	35

Review

Contract Name	DogeBank
Compiler Version	v0.6.8+commit.0bbfe453
Optimization	200 runs
Explorer	https://bscscan.com/address/0x796a6dc7e14e09bf82b4b68be54e9c3e3f231c3d
Address	0x796a6dc7e14e09bf82b4b68be54e9c3e3f231c3d
Network	BSC
Symbol	DogeBank
Decimals	9
Total Supply	1,000,000,000

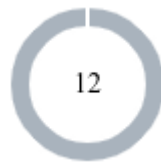
Audit Updates

Initial Audit	28 Mar 2023 https://github.com/cyberscope-io/audits/tree/main/dogeBank/v1/audit.pdf
Corrected Phase 2	30 Mar 2023

Source Files

Filename	SHA256
DogeBank.sol	f72d7a5f6aef983bc90cd19ba246d94324faadc7e7093189f7be2ad97a7b2e82

Findings Breakdown



● Critical	0
● Medium	0
● Minor / Informative	12

Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	0	0	0	0
● Medium	0	0	0	0
● Minor / Informative	12	0	0	0

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OCTD	Transfers Contract's Tokens	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	ULTW	Transfers Liquidity to Team Wallet	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	RPI	Redundant Pair Initializations	Unresolved
●	RC	Redundant Calculations	Unresolved
●	PTAI	Potential Transfer Amount Inconsistency	Unresolved
●	GO	Gas Optimization	Unresolved
●	PAV	Phishing Attack Vulnerability	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

RPI - Redundant Pair Initializations

Criticality	Minor / Informative
Location	contracts/\$DogeBank.sol#L996
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract performs external contract calls on every transfer to retrieve the router's pair address. External calls are expensive in terms of gas consumption. Such operations should be reduced to a minimum.

```
address pancakePair = IPancakeFactory(  
    pancakeRouter.factory()).getPair(address(this), pancakeRouter.WETH());
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it. A recommended approach would be to retrieve the pair address once at the contract's constructor and store it in a global variable, and reuse that variable instead.

RC - Redundant Calculations

Criticality	Minor / Informative
Location	contracts/\$Dogebank.sol#L1011,1030
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The function `paySwapTxFee` adds the fee amount to the `baseAccount` balance and returns the final amount to be transferred to the recipient. As a result, the operations that are performed after the function's call are redundant.

```
paySwapTxFee(  
    from,  
    amount,  
    amount.mul(X2) / 100  
);  
  
uint256 totalFees = amount.mul(X2).div(100);  
_transferStandard(from, to, amount.sub(totalFees));  
...  
paySwapTxFee(  
    from,  
    amount,  
    amount.mul(X3) / 100  
);  
uint256 totalFees = amount.mul(X3).div(100);  
  
_transferStandard(from, to, amount.sub(totalFees));
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it. The contract could use the returning value of the `paySwapTxFee` function, instead of the recalculating the amount.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	contracts/\$Dogebank.sol#L1128
Status	Unresolved

Description

The `transfer()` and `transferFrom()` functions are used to transfer a specified amount of tokens to an address. The fee or tax is an amount that is charged to the sender of an ERC20 token when tokens are transferred to another address. According to the specification, the transferred amount could potentially be less than the expected amount. This may produce inconsistency between the expected and the actual behavior.

The following example depicts the diversion between the expected and actual amount.

Tax	Amount	Expected	Actual
No Tax	100	100	100
10% Tax	100	100	90

The `swapTokensForN` function can either swap tokens for ETH or tokens of another contract to `Dogebank` tokens. The tokens to tokens functionality is implemented by first transferring the user's tokens to the contract's address. If the token supports fees, then the amount that is actually transferred will not be the same as the initial amount. As a result, this produces inconsistency between amounts.

```
IBEP20(tokenAddress).transferFrom(msg.sender, address(this), amount);
```

Recommendation

The team is advised to take into consideration the actual amount that has been transferred instead of the expected.

It is important to note that an ERC20 transfer tax is not a standard feature of the ERC20 specification, and it is not universally implemented by all ERC20 contracts. Therefore, the contract could produce the actual amount by calculating the difference between the transfer call.

```
Actual Transferred Amount = Balance After Transfer - Balance  
Before Transfer
```

GO - Gas Optimization

Criticality	Minor / Informative
Location	contracts/\$DogeBank.sol#L1136,1153,1175,1194
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The functions `swapTokensForN` and `swapNForToken` implement a swapping functionality, where a user can swap tokens for ETH/tokens and tokens for ETH respectively. These functions perform 2 external contract calls each to complete their functionality, and external calls are expensive in terms of gas cost. The same functionality can be achieved with one external call by adding a third address on the `path` variable, the contract's address.

```
pancakeRouter.swapExactTokensForETHSupportingFeeOnTransferTokens(  
    amount,  
    0, // accept any amount of BNB  
    path,  
    address(this),  
    block.timestamp  
);  
  
pancakeRouter.swapExactETHForTokensSupportingFeeOnTransferTokens({value: bnbSwap})(  
    0,  
    path,  
    msg.sender,  
    block.timestamp  
);
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it. As mentioned at the description, the team could add the contract's address to the `path` variable as the third element for the `swapTokensForN` function and the token's address for the `swapNForToken` function. This way, the contract would need to perform only one external call.

PAV - Phishing Attack Vulnerability

Criticality	Minor / Informative
Location	contracts/\$Dogebank.sol#L1004,1083,1088,1094,1100,1106,1222,1228
Status	Unresolved

Description

In a Solidity smart contract, `tx.origin` returns the address of the transaction's sender. However, it is important to note that the sender of the transaction and the user who initiated the transaction can be different.

In the case of a phishing attack, an attacker could send a transaction on behalf of a user to a contract that uses `tx.origin` for authorization. The contract would then grant access to the attacker based on the address returned by `tx.origin`. This is a vulnerability because an attacker can easily spoof the `tx.origin` address by sending a transaction through a malicious contract that modifies the `msg.sender` value.

```
_transferStandard(from, tx.origin, amount);
```

Recommendation

The team is advised to use `msg.sender` instead of `tx.origin` for authorization. The `msg.sender` variable always returns the address of the direct sender of the transaction, which cannot be modified by a malicious contract.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	DogeBank.sol#L876
Status	Unresolved

Description

The contract is using variables that initialize them only in the constructor. The other functions are not mutating the variables. These variables are not defined as `immutable`.

```
pancakeRouter
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	Dogebank.sol#L849,851,852,853
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
uint256 private _tTotal = 1000000000 * 10**9
string private _name = "Dogebank"
string private _symbol = "Dogebank"
uint8 private _decimals = 9
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	DogeBank.sol#L560,562,593,639,867,868,870,871,1064,1076,1082,1087,1093,1099,1105,1111
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
function DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function MINIMUM_LIQUIDITY() external pure returns (uint256);
function WETH() external pure returns (address);
uint256 private X0
uint256 private X1
uint256 private X2
uint256 private X3
uint256 X
address _account
uint256 _amount
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

Find more information on the Solidity documentation

<https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention>.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	Dogebank.sol#L257,286,318,331,350,370,383
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function isContract(address account) internal view returns (bool) {
    // According to EIP-1052, 0x0 is the value returned for not-yet
    created accounts
    // and
    0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned
    // for accounts without code, i.e. `keccak256('')`
    bytes32 codehash;
    bytes32 accountHash =
    0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
    // solhint-disable-next-line no-inline-assembly
    assembly {
        codehash := extcodehash(account)
    }
    return (codehash != accountHash && codehash != 0x0);
}

...
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	DogeBank.sol#L1084
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
baseAccount = _account
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	DogeBank.sol#L264,403
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly {  
    codehash := extcodehash(account)  
}  
  
assembly {  
    let returndata_size := mload(returndata)  
    revert(add(32, returndata), returndata_size)  
}
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	Dogebank.sol#L1128
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
IBEP20(tokenAddress).transferFrom(msg.sender, address(this), amount)
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the [Openzeppelin library](#).

Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
IBEP20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		

Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	_functionCallWithValue	Private	✓	
Ownable	Implementation	Context		
		Internal	✓	
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	geUnlockTime	Public		-
	lock	Public	✓	onlyOwner
	unlock	Public	✓	-
IPancakeFactory	Interface			

	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
IPancakePair	Interface			
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	DOMAIN_SEPARATOR	External		-
	PERMIT_TYPEHASH	External		-
	nonces	External		-
	permit	External	✓	-

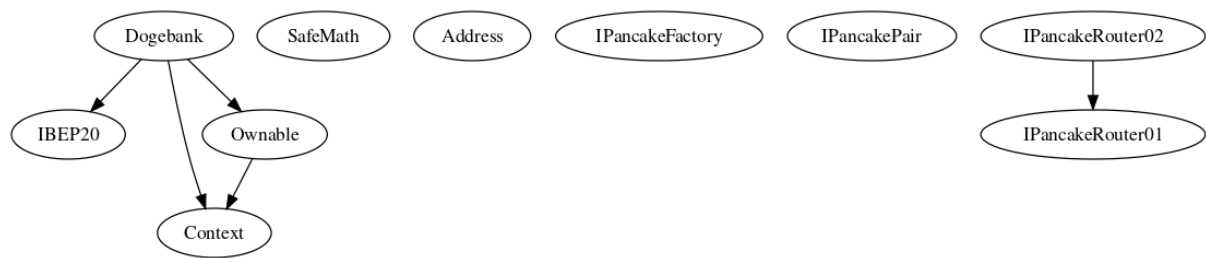
	MINIMUM_LIQUIDITY	External		-
	factory	External		-
	token0	External		-
	token1	External		-
	getReserves	External		-
	price0CumulativeLast	External		-
	price1CumulativeLast	External		-
	kLast	External		-
	mint	External	✓	-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
IPancakeRoute r01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-

	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-
IPancakeRoute r02	Interface	IPancakeRouter01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
DogeBank	Implementation	Context, IBEP20, Ownable		

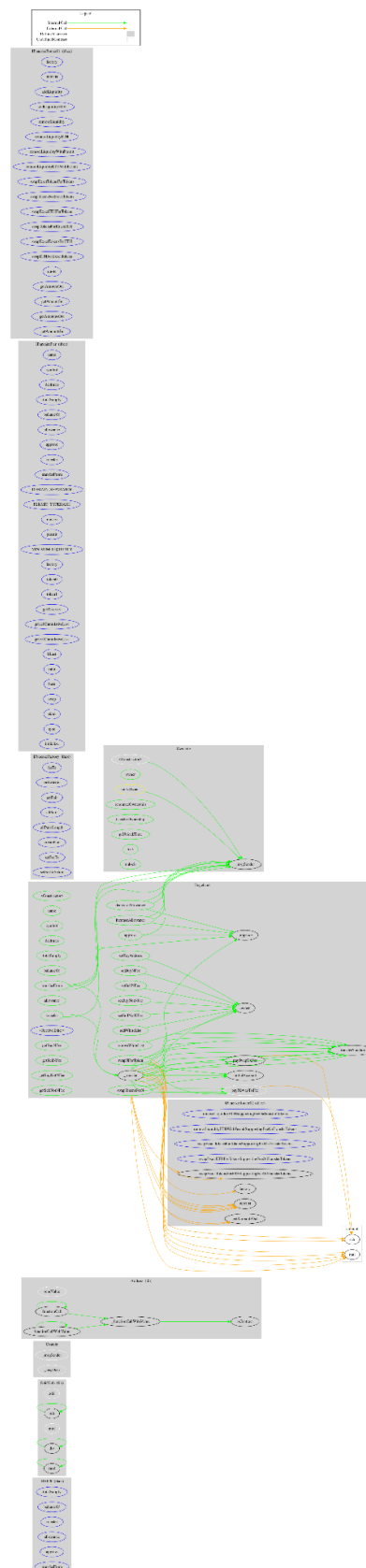
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
		External	Payable	-
	_approve	Private	✓	
	_transfer	Private	✓	
	initialAccount	Internal	✓	
	_transferStandard	Private	✓	
	payNSwapTxFee	Internal	✓	
	paySwapTxFee	Internal	✓	
	setKeyAddress	Public	✓	-
	setBuyNFee	Public	✓	-
	setSellNFee	Public	✓	-
	setBuyNotNFee	Public	✓	-

	setSellNotNFee	Public	✓	-
	swapTokensForN	Public	Payable	-
	swapNForToken	Public	✓	-
	getBuyNFee	Public		-
	getSellNFee	Public		-
	getBuyNotNFee	Public		-
	getSellNotNFee	Public		-
	addWhiteList	Public	✓	-
	removeWhiteList	Public	✓	-

Inheritance Graph



Flow Graph



Summary

Dogebank contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. Dogebank is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions. There is also a limit of max 25% fees.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment 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. Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security. Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.



The Cyberscope team

<https://www.cyberscope.io>



Cyberscope

Audit Report

Dogebank

March 2023

SHA256 46fe90a7f10e7543d6857a85ab02e3493466056e9dfaa7d9aa194789a8d090f0

Audited by © cyberscope

Table of Contents

Table of Contents	1
Review	3
Audit Updates	3
Source Files	3
Findings Breakdown	4
Analysis	5
ELFM - Exceeds Fees Limit	6
Description	6
Recommendation	7
Diagnostics	8
PAV - Phishing Attack Vulnerability	9
Description	9
Recommendation	9
GO - Gas Optimization	10
Description	10
Recommendation	11
PTAI - Potential Transfer Amount Inconsistency	12
Description	12
Recommendation	13
RPI - Redundant Pair Initializations	14
Description	14
Recommendation	14
RC - Redundant Calculations	15
Description	15
Recommendation	16
IDI - Immutable Declaration Improvement	17
Description	17
Recommendation	17
L02 - State Variables could be Declared Constant	18
Description	18
Recommendation	18
L04 - Conformance to Solidity Naming Conventions	19
Description	19
Recommendation	20
L09 - Dead Code Elimination	21
Description	21
Recommendation	22
L16 - Validate Variable Setters	23
Description	23

Recommendation	23
L17 - Usage of Solidity Assembly	24
Description	24
Recommendation	24
L20 - Succeeded Transfer Check	25
Description	25
Recommendation	25
Functions Analysis	26
Inheritance Graph	27
Flow Graph	28
Summary	29
Disclaimer	30
About Cyberscope	31

Review

Contract Name	Dogebank
Testing Deploy	https://testnet.bscscan.com/address/0x2160a9cc99510cfee4bb958b65d14c92207f55aa
Address	0x2160a9cc99510cfee4bb958b65d14c92207f55aa
Symbol	Dogebank
Decimals	9
Total Supply	1,000,000,000

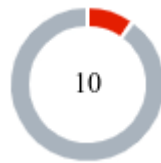
Audit Updates

Initial Audit	28 Mar 2023
---------------	-------------

Source Files

Filename	SHA256
contracts/\$Dogebank.sol	46fe90a7f10e7543d6857a85ab02e3493466056e9dfaa7d9aa194789a8d090f0

Findings Breakdown



Critical	1
Medium	0
Minor / Informative	9

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	0	0	0	0
Minor / Informative	9	0	0	0

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OCTD	Transfers Contract's Tokens	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Unresolved
●	ULTW	Transfers Liquidity to Team Wallet	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	contracts/\$DogeBank.sol#L1099,1105
Status	Unresolved

Description

The contract owner has the authority to increase over the allowed limit of 25%. The owner may take advantage of it by calling the following functions with a high percentage value.

- `setBuyNFee`
- `setSellNFee`
- `setBuyNotNFee`
- `setSellNotNFee`

```
function setBuyNFee(uint256 X) public {
    require(tx.origin == owner());
    require(X + X1 <= 100);
    X0 = X;
}
...
function setSellNFee(uint256 X) public {
    require(tx.origin == owner());
    require(X + X0 <= 100);
    X1 = X;
}
...
function setBuyNotNFee(uint256 X) public {
    require(tx.origin == owner());
    require(X + X3 <= 100);
    X2 = X;
}

function setSellNotNFee(uint256 X) public {
    require(tx.origin == owner());
    require(X + X2 <= 100);
    X3 = X;
}
```

Recommendation

The contract could embody a check for the maximum acceptable value. The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	PAV	Phishing Attack Vulnerability	Unresolved
●	GO	Gas Optimization	Unresolved
●	PTAI	Potential Transfer Amount Inconsistency	Unresolved
●	RPI	Redundant Pair Initializations	Unresolved
●	RC	Redundant Calculations	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

PAV - Phishing Attack Vulnerability

Criticality	Minor / Informative
Location	contracts/\$DogeBank.sol#L1004,1083,1088,1094,1100,1106,1222,1228
Status	Unresolved

Description

In a Solidity smart contract, `tx.origin` returns the address of the transaction's sender. However, it is important to note that the sender of the transaction and the user who initiated the transaction can be different.

In the case of a phishing attack, an attacker could send a transaction on behalf of a user to a contract that uses `tx.origin` for authorization. The contract would then grant access to the attacker based on the address returned by `tx.origin`. This is a vulnerability because an attacker can easily spoof the `tx.origin` address by sending a transaction through a malicious contract that modifies the `msg.sender` value.

```
_transferStandard(from, tx.origin, amount);
```

Recommendation

The team is advised to use `msg.sender` instead of `tx.origin` for authorization. The `msg.sender` variable always returns the address of the direct sender of the transaction, which cannot be modified by a malicious contract.

GO - Gas Optimization

Criticality	Minor / Informative
Location	contracts/\$DogeBank.sol#L1136,1153,1175,1194
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The functions `swapTokensForN` and `swapNForToken` implement a swapping functionality, where a user can swap tokens for ETH/tokens and tokens for ETH respectively. These functions perform 2 external contract calls each to complete their functionality, and external calls are expensive in terms of gas cost. The same functionality can be achieved with one external call by adding a third address on the `path` variable, the contract's address.

```
pancakeRouter.swapExactTokensForETHSupportingFeeOnTransferTokens(  
    amount,  
    0, // accept any amount of BNB  
    path,  
    address(this),  
    block.timestamp  
);  
  
pancakeRouter.swapExactETHForTokensSupportingFeeOnTransferTokens({value: bnbSwap})(  
    0,  
    path,  
    msg.sender,  
    block.timestamp  
);
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it. As mentioned at the description, the team could add the contract's address to the `path` variable as the third element for the `swapTokensForN` function and the token's address for the `swapNForToken` function. This way, the contract would need to perform only one external call.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	contracts/\$DogeBank.sol#L1128
Status	Unresolved

Description

The `transfer()` and `transferFrom()` functions are used to transfer a specified amount of tokens to an address. The fee or tax is an amount that is charged to the sender of an ERC20 token when tokens are transferred to another address. According to the specification, the transferred amount could potentially be less than the expected amount. This may produce inconsistency between the expected and the actual behavior.

The following example depicts the diversion between the expected and actual amount.

Tax	Amount	Expected	Actual
No Tax	100	100	100
10% Tax	100	100	90

The `swapTokensForN` function can either swap tokens for ETH or tokens of another contract to `DogeBank` tokens. The tokens to tokens functionality is implemented by first transferring the user's tokens to the contract's address. If the token supports fees, then the amount that is actually transferred will not be the same as the initial amount. As a result, this produces inconsistency between amounts.

```
IBEP20(tokenAddress).transferFrom(msg.sender, address(this), amount);
```

Recommendation

The team is advised to take into consideration the actual amount that has been transferred instead of the expected.

It is important to note that an ERC20 transfer tax is not a standard feature of the ERC20 specification, and it is not universally implemented by all ERC20 contracts. Therefore, the contract could produce the actual amount by calculating the difference between the transfer call.

```
Actual Transferred Amount = Balance After Transfer - Balance  
Before Transfer
```

RPI - Redundant Pair Initializations

Criticality	Minor / Informative
Location	contracts/\$DogeBank.sol#L996
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract performs external contract calls on every transfer to retrieve the router's pair address. External calls are expensive in terms of gas consumption. Such operations should be reduced to a minimum.

```
address pancakePair = IPancakeFactory(  
    pancakeRouter.factory()).getPair(address(this), pancakeRouter.WETH());
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it. A recommended approach would be to retrieve the pair address once at the contract's constructor and store it in a global variable, and reuse that variable instead.

RC - Redundant Calculations

Criticality	Minor / Informative
Location	contracts/\$Dogebank.sol#L1011,1030
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The function `paySwapTxFee` adds the fee amount to the `baseAccount` balance and returns the final amount to be transferred to the recipient. As a result, the operations that are performed after the function's call are redundant.

```
paySwapTxFee(  
    from,  
    amount,  
    amount.mul(X2) / 100  
);  
  
uint256 totalFees = amount.mul(X2).div(100);  
_transferStandard(from, to, amount.sub(totalFees));  
...  
paySwapTxFee(  
    from,  
    amount,  
    amount.mul(X3) / 100  
);  
uint256 totalFees = amount.mul(X3).div(100);  
  
_transferStandard(from, to, amount.sub(totalFees));
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it. The contract could use the returning value of the `paySwapTxFee` function, instead of recalculating the amount.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/\$DogeBank.sol#L876
Status	Unresolved

Description

The contract is using variables that initialize them only in the constructor. The other functions are not mutating the variables. These variables are not defined as `immutable`.

```
pancakeRouter
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/\$Dogebank.sol#L849,851,852,853
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
uint256 private _tTotal = 1000000000 * 10**9
string private _name = "Dogebank"
string private _symbol = "Dogebank"
uint8 private _decimals = 9
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/\$Dogebank.sol#L560,562,593,639,867,868,870,871,1064,1076,1082,1087,1093,1099,1105,1111
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
function DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function MINIMUM_LIQUIDITY() external pure returns (uint256);
function WETH() external pure returns (address);
uint256 private X0
uint256 private X1
uint256 private X2
uint256 private X3
uint256 X
address _account
uint256 _amount
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

Find more information on the Solidity documentation

<https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention>.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	contracts/\$DogeBank.sol#L257,286,318,331,350,370,383
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function isContract(address account) internal view returns (bool) {
    // According to EIP-1052, 0x0 is the value returned for not-yet
    created accounts
    // and
    0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned
    // for accounts without code, i.e. `keccak256('')`
    bytes32 codehash;
    bytes32 accountHash =
    0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
    // solhint-disable-next-line no-inline-assembly
    assembly {
        codehash := extcodehash(account)
    }
    return (codehash != accountHash && codehash != 0x0);
}

...
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/\$DogeBank.sol#L1084
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
baseAccount = _account
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	contracts/\$DogeBank.sol#L264,403
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly {  
    codehash := extcodehash(account)  
}  
  
assembly {  
    let returndata_size := mload(returndata)  
    revert(add(32, returndata), returndata_size)  
}
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	contracts/\$Dogebank.sol#L1128
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
IBEP20(tokenAddress).transferFrom(msg.sender, address(this), amount)
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the [Openzeppelin library](#).

Functions Analysis

Inheritance Graph

Flow Graph

Summary

Token is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions. There is also a limit of max 25% fees.

There are some functions that can be abused by the owner, like manipulating fees and transferring funds to the team's wallet. The maximum fee percentage that can be set is 25%. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

The contract can be converted into a honeypot and prevent users from selling if the owner abuses the admin functions.

This audit investigates security issues, business logic concerns and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment 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. Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security. Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.



The Cyberscope team

<https://www.cyberscope.io>