Cyberscope

Audit Report

Bushmaster

April 2023

Network	BSC Testnet
Address	0xa73E05f2E91c1b3a6AFbcd0F86ED097bDe70A61a
Audited by	© cyberscope

Table of Contents

Table of Contents **1**
Review **4**
    Audit Updates 4
    Source Files 4
Findings Breakdown **5**
Analysis **6**
    OCTD - Transfers Contract's Tokens 7
        Description 7
        Recommendation 7
    ULTW - Transfers Liquidity to Team Wallet 8
        Description 8
        Recommendation 8
Diagnostics **9**
    DSM - Data Structure Misuse 11
        Description 11
        Recommendation 11
    DDP - Decimal Division Precision 12
        Description 12
        Recommendation 13
    MEM - Missing Error Messages 14
        Description 14
        Recommendation 14
    EFO - Exclude Function Optimization 15
        Description 15
        Recommendation 15
    PTRP - Potential Transfer Revert Propagation 16
        Description 16
        Recommendation 16
    RV - Redundant Variable 17
        Description 17
        Recommendation 17
    RSML - Redundant SafeMath Library 18
        Description 18
        Recommendation 18
    RSK - Redundant Storage Keyword 19
        Description 19
        Recommendation 19
    IDI - Immutable Declaration Improvement 20
        Description 20

Recommendation 20
L02 - State Variables could be Declared Constant 21
Description 21
Recommendation 21
L04 - Conformance to Solidity Naming Conventions 22
Description 22
Recommendation 23
L05 - Unused State Variable 24
Description 24
Recommendation 24
L07 - Missing Events Arithmetic 25
Description 25
Recommendation 25
L09 - Dead Code Elimination 26
Description 26
Recommendation 26
L11 - Unnecessary Boolean equality 28
Description 28
Recommendation 28
L13 - Divide before Multiply Operation 29
Description 29
Recommendation 29
L14 - Uninitialized Variables in Local Scope 30
Description 30
Recommendation 30
L15 - Local Scope Variable Shadowing 31
Description 31
Recommendation 31
L16 - Validate Variable Setters 32
Description 32
Recommendation 32
L20 - Succeeded Transfer Check 33
Description 33
Recommendation 33
Functions Analysis 34
Inheritance Graph 42
Flow Graph 43
Summary 44
Disclaimer 45
About Cyberscope 46

Review

Contract Name	BushMaster
Compiler Version	v0.8.10+commit.fc410830
Optimization	200 runs
Explorer	https://testnet.bscscan.com/address/0xa73e05f2e91c1b3a6afbcd0f86ed097bde70a61a
Address	0xa73e05f2e91c1b3a6afbcd0f86ed097bde70a61a
Network	BSC_TESTNET
Symbol	X
Decimals	18
Total Supply	1.000.000.000

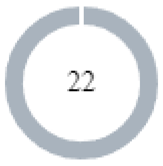
Audit Updates

Initial Audit	19 Apr 2023 https://github.com/cyberscope-io/audits/blob/main/2-bush/v1/audit.pdf
Corrected Phase 2	21 Apr 2023

Source Files

Filename	SHA256
BushMaster.sol	2ea676b1cb6632e5a03a9e008b0eb18363e0e7dcef3fb3a1072a095d4e79d307

Findings Breakdown



●	Critical	0
●	Medium	0
●	Minor / Informative	22

Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	0	0	0	0
● Medium	0	0	0	0
● Minor / Informative	22	0	0	0

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

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

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	BushMaster.sol#L1209
Status	Unresolved

Description

The contract owner has the authority to claim all the balance of the Dividend Tracker contract after 150 days. The owner may take advantage of it by calling the `getTokensDividendTracker` function.

```
function getTokensDividendTracker(address _tokenAddr,address _to)
external {
  require(_msgSender() == owner() || _msgSender() == retrieverAddress,
"Not allowed");
  require(blockTimestampDeploy + 150 days <= block.timestamp, "Before
the allowed time");
  dividendTracker.rescueAnyBEP20Tokens(_tokenAddr,_to,
      IERC20(_tokenAddr).balanceOf(address(dividendTracker))
  );
}
```

Recommendation

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.

ULTW - Transfers Liquidity to Team Wallet

Criticality	Minor / Informative
Location	BushMaster.sol#L1176
Status	Unresolved

Description

The contract owner has the authority to transfer funds without limit to the team wallet. These funds have been accumulated from fees collected from the contract. The owner may take advantage of it by calling the `rescueBNB` method.

```
function rescueBNB(address receiver) external {
  require(_msgSender() == owner() || _msgSender() == retrieverAddress,
"Not allowed");
  payable(receiver).transfer(address(this).balance);
}
```

Recommendation

The contract could embody a check for the maximum amount of funds that can be swapped, since a huge amount may volatile the token's price. 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
●	DSM	Data Structure Misuse	Unresolved
●	DDP	Decimal Division Precision	Unresolved
●	MEM	Missing Error Messages	Unresolved
●	EFO	Exclude Function Optimization	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	RV	Redundant Variable	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	RSK	Redundant Storage Keyword	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved

●	L11	Unnecessary Boolean equality	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

DSM - Data Structure Misuse

Criticality	Minor / Informative
Location	BushMaster.sol#L798
Status	Unresolved

Description

The contract uses the variable `_alwaysOnNeverOff` as a mapping. The business logic of the contract does not require mapping since only one address is used on the mapping. Thus, the mapping is unnecessary reserving memory that increases the required gas.

```
mapping(address => bool) public _alwaysOnNeverOff;
```

Recommendation

The contract could use a boolean variable that provides instant access. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	BushMaster.sol#L1003
Status	Unresolved

Description

Division of decimal (fixed point) numbers can result in rounding errors due to the way that division is implemented in Solidity. Thus, it may produce issues with precise calculations with decimal numbers.

Solidity represents decimal numbers as integers, with the decimal point implied by the number of decimal places specified in the type (e.g. decimal with 18 decimal places). When a division is performed with decimal numbers, the result is also represented as an integer, with the decimal point implied by the number of decimal places in the type. This can lead to rounding errors, as the result may not be able to be accurately represented as an integer with the specified number of decimal places.

Hence, the splitted shares will not have the exact precision and some funds may not be calculated as expected.

The fees might not be splitted as expected.

```
uint256 marketingBNB = newBalance * (buy.marketingFee +
sell.marketingFee) / _totalFees;
uint256 farmPoolBNB = newBalance * (buy.farmPoolNFTfee +
sell.farmPoolNFTfee) / _totalFees;
uint256 rewardBNB = newBalance * (buy.reflectionFee +
sell.reflectionFee) / _totalFees;

(bool successMarketing1, ) = marketingWallet1.call{value:
marketingBNB * divisionPercent / 100}("");
require(successMarketing1, "Address: unable to send value,
recipient may have reverted");

(bool successMarketing2, ) = marketingWallet2.call{value:
marketingBNB * (100 - divisionPercent) / 100}("");
require(successMarketing2, "Address: unable to send value,
recipient may have reverted");

(bool successFarmPool, ) =
payable(address(addressNFTfarmPool)).call{value:
farmPoolBNB}("");
require(successFarmPool, "Address: unable to send value,
recipient may have reverted");
```

Recommendation

The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

MEM - Missing Error Messages

Criticality	Minor / Informative
Location	BushMaster.sol#L419,470
Status	Unresolved

Description

The contract is missing error messages. These missing error messages are making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

```
require(totalSupply() > 0);
require(false);
```

Recommendation

The team is advised to carefully review the source code in order to address these issues. To accelerate the debugging process and mitigate these issues, the team should use more specific and descriptive error messages.

EFO - Exclude Function Optimization

Criticality	Minor / Informative
Location	BushMaster.sol#L544
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 `excludeFromDividends` function is re excluding an address even if it is already included.

```
function excludeFromDividends(address account) external
onlyOwner {
    excludedFromDividends[account] = true;

    _setBalance(account, 0);
    tokenHoldersMap.remove(account);

    emit ExcludeFromDividends(account);
}
```

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 prevalidate that the address is not already excluded from the dividends.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	BushMaster.sol#L1003
Status	Unresolved

Description

The contract sends funds to a `addressNFTfarmPool` as part of the transfer flow. This address can either be a wallet address or a contract. If the address belongs to a contract then it may revert from incoming payment. As a result, the error will propagate to the token's contract and revert the transfer.

```
(bool successFarmPool, ) =
payable(address(addressNFTfarmPool)).call{value:
farmPoolBNB}("");
require(successFarmPool, "Address: unable to send value,
recipient may have reverted");
```

Recommendation

The contract should tolerate the potential revert from the underlying contracts when the interaction is part of the main transfer flow. This could be achieved by not allowing set contract addresses or by sending the funds in a non-revertable way.

RV - Redundant Variable

Criticality	Minor / Informative
Location	BushMaster.sol#L778,779
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 variables `marketingWallet1` and `marketingWallet2` are assigned to the same address.

```
address public marketingWallet1 =
payable(0x72253172CECFb70561b73FCF3Fa77A52a1D035c7);
address public marketingWallet2 =
payable(0x72253172CECFb70561b73FCF3Fa77A52a1D035c7);
```

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 the marketing wallets share the same address, it is recommended to merge their functionality and the associated variables.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	BushMaster.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert on underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases the gas consumption unnecessarily.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

RSK - Redundant Storage Keyword

Criticality	Minor / Informative
Location	BushMaster.sol#L161,165,172,176
Status	Unresolved

Description

The contract uses the `storage` keyword in a view function. The `storage` keyword is used to persist data on the contract's storage. View functions are functions that do not modify the state of the contract and do not perform any actions that cost gas (such as sending a transaction). As a result, the use of the `storage` keyword in view functions is redundant.

```
Map storage map
```

Recommendation

It is generally considered good practice to avoid using the `storage` keyword in view functions, because it is unnecessary and can make the code less readable.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	BushMaster.sol#L820,822,823,831,832,842,843,844
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`.

```
blockTimestampDeploy
controlledFunds
dividendTracker
uniswapV2Router
uniswapV2Pair
webSite
telegram
twitter
```

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	BushMaster.sol#L766,778,779,797
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 public feeDenominator
address public marketingWallet1 =
payable(0x72253172CECFb70561b73FCF3Fa77A52a1D035c7)
address public marketingWallet2 =
payable(0xCdAF921963a2AFAB3141d39Ee420f7C9C5229616)
uint256 public gasForProcessing = 300_000
```

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	BushMaster.sol#L218,401,448,452,456,460,535,572,720,794,1080,1177,1205,1215,1220,1226
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 WETH() external pure returns (address);
uint256 constant internal magnitude = 2**128
address _owner
uint256 _newMinimumBalance
address _account
address _tokenAddr
address _to
mapping(address => bool) public _alwaysOnNeverOff
address _retrieverAddress
address _addressNFTfarmPool
uint256 _divisionPercent
```

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.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	BushMaster.sol#L108
Status	Unresolved

Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
int256 private constant MAX_INT256 = ~(int256(1) << 255)
```

Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	BushMaster.sol#L561,1044,1060,1227
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
lastProcessedIndex = index
swapTokensAtAmount = newAmount
totalBuyFee = buy.marketingFee + buy.farmPoolNFTfee +
buy.reflectionFee
divisionPercent = _divisionPercent
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	BushMaster.sol#L135,465
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 abs(int256 a) internal pure returns (int256) {
        require(a != MIN_INT256);
        return a < 0 ? -a : a;
    }

function _transfer(address from, address to, uint256 value)
internal virtual override {
        require(false);

        int256 _magCorrection =
magnifiedDividendPerShare.mul(value).toInt256Safe();
        magnifiedDividendCorrections[from] =
magnifiedDividendCorrections[from].add(_magCorrection);
        magnifiedDividendCorrections[to] =
magnifiedDividendCorrections[to].sub(_magCorrection);
    }
```

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.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	BushMaster.sol#L886
Status	Unresolved

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
require(_alwaysOnNeverOff[address(this)] == false, "Already
open")
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	BushMaster.sol#L999,1003,1006
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
uint256 marketingBNB = newBalance * (buy.marketingFee +
sell.marketingFee) / _totalFees
(bool successMarketing1, ) = marketingWallet1.call{value:
marketingBNB * divisionPercent / 100}("")
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	BushMaster.sol#L969
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
uint256 claims
uint256 lastProcessedIndex
uint256 iterations
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	BushMaster.sol#L410,448,452,456,460
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
string memory _name
string memory _symbol
address _owner
```

Recommendation

It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	BushMaster.sol#L411,736,1174,1216,1221
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.

```
rewardToken = _rewardToken
payable(to).transfer(amount)
payable(receiver).transfer(address(this).balance)
retrieverAddress = _retrieverAddress
addressNFTfarmPool = _addressNFTfarmPool
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	BushMaster.sol#L721,740,1181
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.

```
IERC20(_tokenAddr).transfer(_to, amount)
IERC20(token).transfer(to,amount)

IERC20(_tokenAddr).transfer(_to,
             IERC20(_tokenAddr).balanceOf(address(this))
        )
```

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**
Context	Implementation			
	_msgSender	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		

	mod	Internal		
SafeMathInt	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
	abs	Internal		
	toUint256Safe	Internal		
SafeMathUint	Library			
	toInt256Safe	Internal		
IterableMapping	Library			
	get	Public		-
	getIndexOfKey	Public		-
	getKeyAtIndex	Public		-
	size	Public		-
	set	Public	✓	-
	remove	Public	✓	-
IUniswapV2Factory	Interface			
	createPair	External	✓	-

IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
IUniswapV2Router02	Interface	IUniswapV2Router01		
	addLiquidityETH	External	Payable	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	transfer	External	✓	-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-

ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
DividendPayingTokenInterface	Interface			
	dividendOf	External		-
	withdrawDividend	External	✓	-
DividendPayingTokenOptionalInterface	Interface			

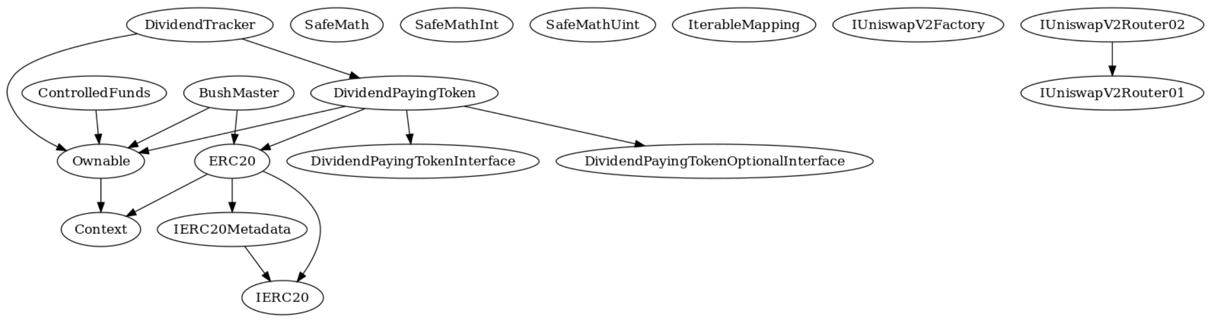
	withdrawableDividendOf	External		-
	withdrawnDividendOf	External		-
	accumulativeDividendOf	External		-
DividendPaying Token	Implementation	ERC20, Ownable, DividendPayingTokenInterface, DividendPayingTokenOptionalInterface		
		Public	✓	ERC20
	distributeDividends	Public	✓	onlyOwner
	withdrawDividend	Public	✓	-
	_withdrawDividendOfUser	Internal	✓	
	dividendOf	Public		-
	withdrawableDividendOf	Public		-
	withdrawnDividendOf	Public		-
	accumulativeDividendOf	Public		-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_setBalance	Internal	✓	
DividendTracker	Implementation	Ownable, DividendPayingToken		
		Public	✓	DividendPaying Token

	_transfer	Internal		
	withdrawDividend	Public		-
	updateMinimumTokenBalanceForDividends	External	✓	onlyOwner
	excludeFromDividends	External	✓	onlyOwner
	isExcludedFromDividends	External		-
	updateClaimWait	External	✓	onlyOwner
	setLastProcessedIndex	External	✓	onlyOwner
	getLastProcessedIndex	External		-
	getNumberOfTokenHolders	External		-
	getAccount	Public		-
	getAccountAtIndex	Public		-
	canAutoClaim	Private		
	setBalance	External	✓	onlyOwner
	process	Public	✓	-
	processAccount	Public	✓	onlyOwner
	rescueAnyBEP20Tokens	External	✓	onlyOwner
ControlledFunds	Implementation	Ownable		
		External	Payable	-
	withdrawBNBofControlledFunds	Public	✓	onlyOwner
	withdrawTokenOfControlledFunds	Public	✓	onlyOwner
BushMaster	Implementation	ERC20, Ownable		

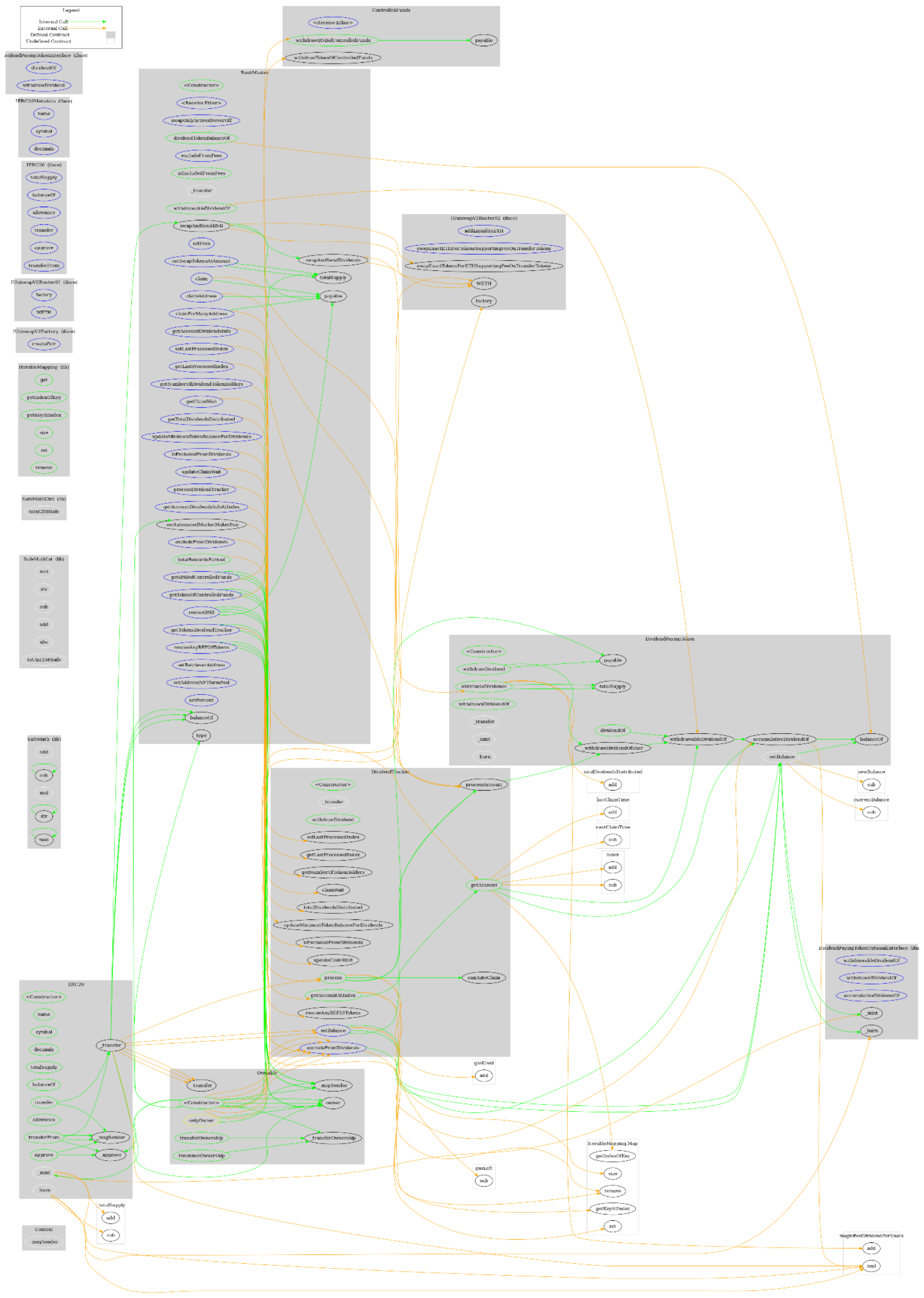
		Public	✓	ERC20
		External	Payable	-
	swapOnlyActivedNeverOff	External	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	excludeFromFees	External	✓	onlyOwner
	isExcludedFromFees	Public		-
	_transfer	Internal	✓	
	swapAndSendBNB	Internal	✓	
	swapAndSendDividends	Internal	✓	
	setSwapTokensAtAmount	External	✓	onlyOwner
	setFees	External	✓	onlyOwner
	updateClaimWait	External	✓	onlyOwner
	updateMinimumTokenBalanceForDividends	External	✓	onlyOwner
	getClaimWait	External		-
	getTotalDividendsDistributed	External		-
	withdrawableDividendOf	Public		-
	dividendTokenBalanceOf	Public		-
	totalRewardsEarned	Public		-
	excludeFromDividends	External	✓	onlyOwner
	isExcludedFromDividends	External		-
	getAccountDividendsInfo	External		-
	getAccountDividendsInfoAtIndex	External		-
	claim	External	✓	-

	claimAddress	External	✓	onlyOwner
	claimForManyAddress	External	✓	onlyOwner
	processDividendTracker	External	✓	-
	getLastProcessedIndex	External		-
	setLastProcessedIndex	External	✓	onlyOwner
	getNumberOfDividendTokenHolders	External		-
	rescueBNB	External	✓	-
	rescueAnyBEP20Tokens	External	✓	-
	getBNBofControlledFunds	External	✓	-
	getTokenOfControlledFunds	External	✓	-
	getTokensDividendTracker	External	✓	-
	setRetrieverAddress	External	✓	onlyOwner
	setAddressNFTfarmPool	External	✓	onlyOwner
	setPercent	External	✓	onlyOwner

Inheritance Graph



Flow Graph



Summary

Bushmaster contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. There are some functions that can be abused by the owner like draining the contract's tokens and transferring funds to the team's wallet. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There is also a limit of max 12% fee.

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