Cyberscope

Audit Report

Elons Roadmap

May 2023

Network	BSC
Address	0x73d37a93a95d4bC43017aaB63Da4d499b49B0F2C
Audited by	© cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

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

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	MMN	Misleading Modifier Naming	Unresolved
●	RVD	Redundant Variable Declaration	Unresolved
●	RSD	Redundant Struct Declaration	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	ElonsRoadmap
Compiler Version	v0.8.0+commit.c7dfd78e
Optimization	200 runs
Explorer	https://bscscan.com/address/0x73d37a93a95d4bc43017aab63da4d499b49b0f2c
Address	0x73d37a93a95d4bc43017aab63da4d499b49b0f2c
Network	BSC
Symbol	ELMAP
Decimals	9
Total Supply	1,000,000,000

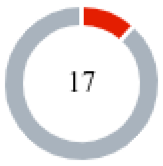
Audit Updates

Initial Audit	28 May 2023

Source Files

Filename	SHA256
ElonsRoadmap.sol	3ff0beec8a6a0467b8bf2d708240b163adcba8342299df7572c45fd68aa65cfc

Findings Breakdown



●	Critical	2
●	Medium	0
●	Minor / Informative	15

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	2	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	15	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	ElonsRoadmap.sol#L864,876
Status	Unresolved

Description

The contract owner has the authority to stop the transactions for all users excluding the premarket addresses. The owner may take advantage of it by setting the `marketActive` to false.

```
if(!marketActive) {
    require(premarketUser[from],"cannot trade before the market opening");
}
```

The contract owner has the authority to stop the transactions for all users excluding the owner. The owner may take advantage of it by setting the `maxBuyTxAmount` to zero.

```
if(limitBuys) {
    require(amount <= maxBuyTxAmount, "maxBuyTxAmount Limit Exceeded");
}
```

Recommendation

The contract could embody a check for not allowing setting the `maxBuyTxAmount` less than a reasonable amount. A suggested implementation could check that the minimum amount should be more than a fixed percentage of the total supply. 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.

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	ElonsRoadmap.sol#L572,577,582,587
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 either of the following functions with a high percentage value:

- `setReflectionFee`
- `setDevelopFee`
- `setLiquidityFee`
- `setMarketingFee`

```
function setReflectionFee(uint buy, uint sell) external onlyOwner() {
    buyReflectionFee = buy;
    sellReflectionFee = sell;
    setFees();
}
function setDevelopFee(uint buy, uint sell) external onlyOwner() {
    buyDevelopFee = buy;
    sellDevelopFee = sell;
    setFees();
}
function setLiquidityFee(uint buy, uint sell) external onlyOwner() {
    buyLiqFee = buy;
    sellLiqFee = sell;
    setFees();
}
function setMarketingFee(uint buy, uint sell) external onlyOwner() {
    buyMarketingFee = buy;
    sellMarketingFee = sell;
    setFees();
}
```

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.

ULTW - Transfers Liquidity to Team Wallet

Criticality	Minor / Informative
Location	ElonsRoadmap.sol#L800
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 `Sweep` method.

```
function Sweep() external onlyOwner {
    uint balance = address(this).balance;
    payable(owner()).transfer(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.

MMN - Misleading Modifier Naming

Criticality	Minor / Informative
Location	ElonsRoadmap.sol#L758
Status	Unresolved

Description

Modifiers can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The contract uses some modifier names that are too generic or do not clearly convey the underneath functionality. Misleading modifier names can lead to confusion, making the code more difficult to read and understand. modifiers can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The contract uses some modifier names that are too generic or do not clearly convey the underneath functionality. Misleading modifier names can lead to confusion, making the code more difficult to read and understand.

The `FastTx` modifier is executed when the contract is swapping tokens. Hence, its name does not reflect its functionality.

```
modifier FastTx() {
    isInternalTransaction = true;
    _;
    isInternalTransaction = false;
}
```

Recommendation

It's always a good practice for the contract to contain modifier names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

RVD - Redundant Variable Declaration

Criticality	Minor / Informative
Location	ElonsRoadmap.sol#L378,402
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 declares some variables that are not used in a meaningful way by the contract. As a result, these variables are redundant.

```
bool public TakeBnbForFees = true;
uint public maxSellTxAmount;
```

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.

RSD - Redundant Struct Declaration

Criticality	Minor / Informative
Location	ElonsRoadmap.sol#L419,431
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 declares the `userData` struct to keep track of each user's last buy. Since the struct only contains one property, it could be omitted. As a result the struct is redundant.

```
struct userData {
    uint lastBuyTime;
}
mapping (address => userData) public userLastTradeData;
```

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 modify the `userLastTradeData` mapping to return a uint256 integer for each address instead of a struct.

```
mapping (address => uint256) public userLastTradeData;
```

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	ElonsRoadmap.sol#L830
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `minimumTokensBeforeSwap` sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then the contract will swap a huge amount of tokens for ETH.

It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly at the triggered point, potentially leading to significant price volatility for the parties involved.

```
function setSwapAndLiquify(bool _state, uint _minimumTokensBeforeSwap)
external onlyOwner {
    swapAndLiquifyEnabled = _state;
    minimumTokensBeforeSwap = _minimumTokensBeforeSwap;
}
```

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount should be less than a fixed percentage of the total supply. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	ElonsRoadmap.sol#L472,557,561,564,805,811,814,818,821,835,838,841
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 modifies the state of some variables without checking if the current state of these variables is the same as the one given as an argument. As a result, the contract performs redundant storage writes.

```
function setMoveBnbToWallets(bool state) external onlyOwner {
    moveBnbToWallets = state;
}

function excludeFromFee(address account) public onlyOwner {
    excludedFromFees[account] = true;
}

function includeInFee(address account) public onlyOwner {
    excludedFromFees[account] = false;
}
...
```

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.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	ElonsRoadmap.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.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	ElonsRoadmap.sol#L438,447
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`.

```
uniswapV2Router
uniswapV2Pair
```

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	ElonsRoadmap.sol#L372,373,378,400,406,417
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.

```
string private _name = "Elons Roadmap"
string private _symbol = "ELMAP"
bool public TakeBnbForFees = true
uint public buySecondsLimit = 5
uint private _tTotal = 1000000000 * (10**9)
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	ElonsRoadmap.sol#L288,378,386,387,404,409,410,411,412,413,414,415,416,419,665,671,677,682,722,758,793,800,805,811,814,818,821,824,827,830,835,838,841
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);
bool public TakeBnbForFees = true
address public MarketingWallet_Address = 0xA92C57b0C9ede3c9B7CA7334eC46DF1177D7bb63
address public DeveloperWallet_Address = 0xA92C57b0C9ede3c9B7CA7334eC46DF1177D7bb63
uint private MarketActiveAt
uint private _ReflectionFee
uint private _DevelopFee
uint private _LiqFee
uint private _MarketingFee
uint private _OldReflectionFee
uint private _OldDevelopFee
uint private _OldLiqFee
uint private _OldMarketingFee

...
```

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	ElonsRoadmap.sol#L573,578,583,588,593,828,832
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.

```
buyReflectionFee = buy
buyDevelopFee = buy
buyLiqFee = buy
buyMarketingFee = buy
maxBuyTxAmount = (_tTotal * buy) / 10**2
maxBuyTxAmount = _value
minimumTokensBeforeSwap = _minimumTokensBeforeSwap
```

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	ElonsRoadmap.sol#L21,27,35,39,43,47,56,60,68,72,80
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) {
        uint256 size;
        // solhint-disable-next-line no-inline-assembly
        assembly { size := extcodesize(account) }
        return size > 0;
    }

function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
        (bool success, ) = recipient.call{ value: amount }("");
        require(success, "Address: unable to send value, recipient may have
reverted");
    }

...
```

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	ElonsRoadmap.sol#L819,822
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.

```
MarketingWallet_Address = _value
DeveloperWallet_Address = _value
```

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	ElonsRoadmap.sol#L24,89
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 { size := extcodesize(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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	ElonsRoadmap.sol#L18
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.0;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	_verifyCallResult	Private		
SafeMath	Library			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		

	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		
	sub	Internal		
	div	Internal		
	mod	Internal		
IUniswapV2Factory	Interface			
	createPair	External	✓	-
IUniswapV2Router02	Interface			
	factory	External		-
	WETH	External		-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
	addLiquidityETH	External	Payable	-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-

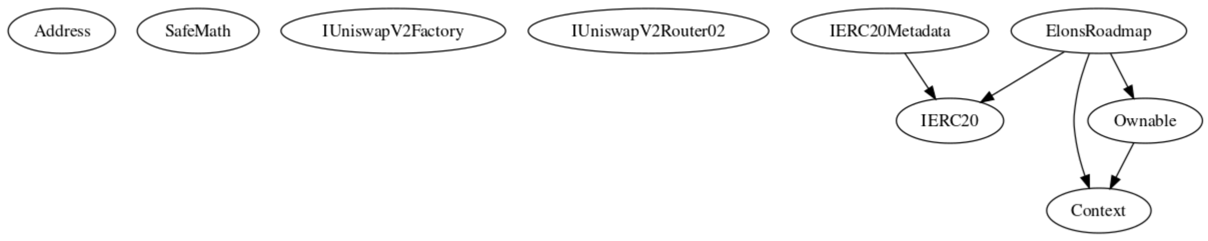
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	getTime	Public		-
ElonsRoadmap	Implementation	Context, IERC20, Ownable		
		Public	✓	-
		External	Payable	-

	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	setMoveBnbToWallets	External	✓	onlyOwner
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	isExcludedFromReward	Public		-
	totalFees	Public		-
	reflectionFromToken	Public		-
	tokenFromReflection	Public		-
	excludeFromReward	Public	✓	onlyOwner
	includeInReward	External	✓	onlyOwner
	excludeFromFee	Public	✓	onlyOwner
	includeInFee	Public	✓	onlyOwner
	setSwap	External	✓	onlyOwner
	setFees	Private	✓	
	setReflectionFee	External	✓	onlyOwner

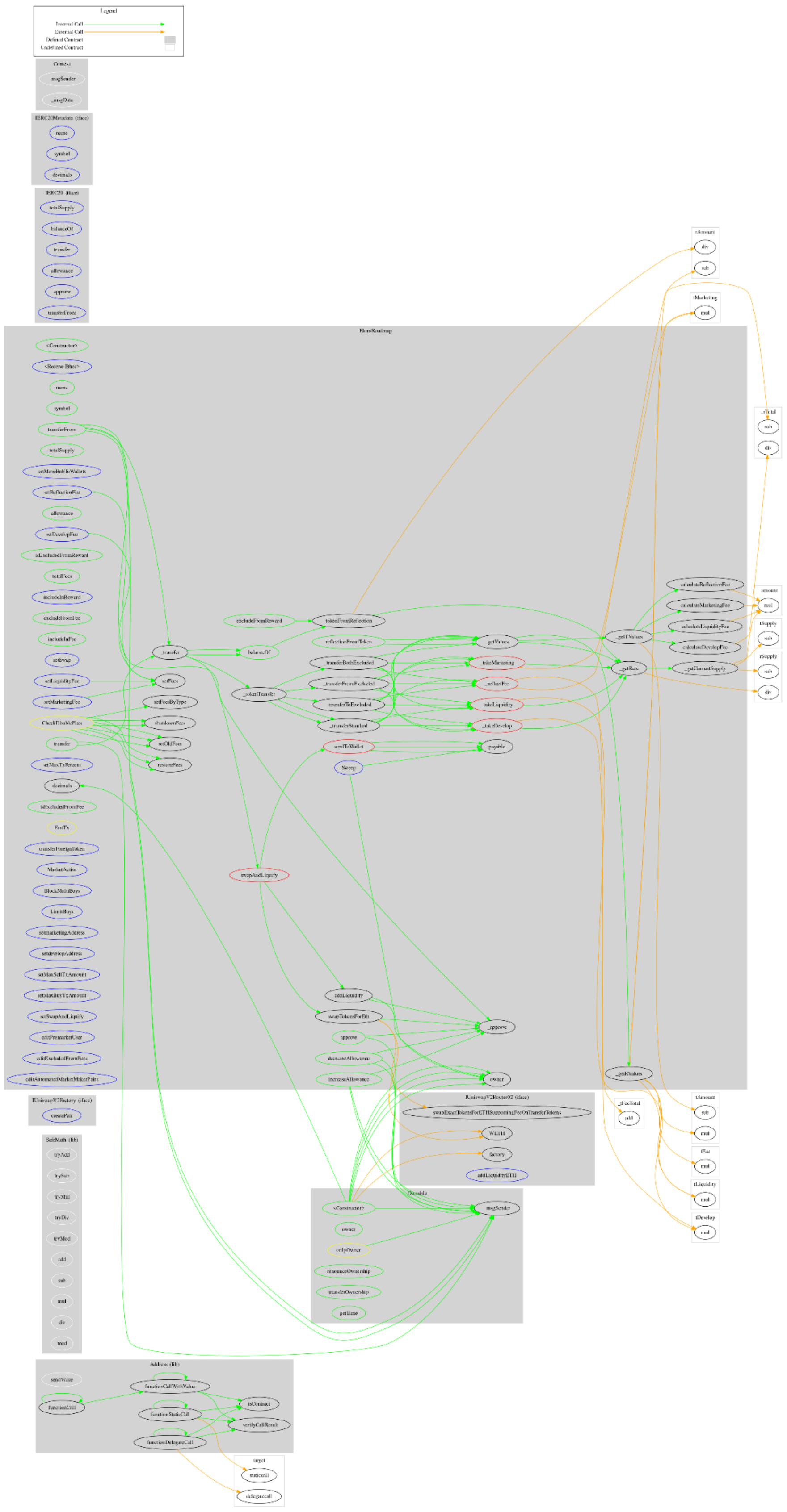
	setDevelopFee	External	✓	onlyOwner
	setLiquidityFee	External	✓	onlyOwner
	setMarketingFee	External	✓	onlyOwner
	setMaxTxPercent	External	✓	onlyOwner
	_reflectFee	Private	✓	
	_getValues	Private		
	_getTValues	Private		
	_getRValues	Private		
	_getRate	Private		
	_getCurrentSupply	Private		
	_takeLiquidity	Private	✓	
	_takeDevelop	Private	✓	
	_takeMarketing	Private	✓	
	calculateReflectionFee	Private		
	calculateDevelopFee	Private		
	calculateLiquidityFee	Private		
	calculateMarketingFee	Private		
	setOldFees	Private	✓	
	shutdownFees	Private	✓	
	setFeesByType	Private	✓	
	restoreFees	Private	✓	
	isExcludedFromFee	Public		-
	_approve	Private	✓	

	sendToWallet	Private	✓	
	swapAndLiquify	Private	✓	FastTx
	transferForeignToken	External	✓	onlyOwner
	Sweep	External	✓	onlyOwner
	MarketActive	External	✓	onlyOwner
	BlockMultiBuys	External	✓	onlyOwner
	LimitBuys	External	✓	onlyOwner
	setmarketingAddress	External	✓	onlyOwner
	setdevelopAddress	External	✓	onlyOwner
	setMaxSellTxAmount	External	✓	onlyOwner
	setMaxBuyTxAmount	External	✓	onlyOwner
	setSwapAndLiquify	External	✓	onlyOwner
	editPremarketUser	External	✓	onlyOwner
	editExcludedFromFees	External	✓	onlyOwner
	editAutomatedMarketMakerPairs	External	✓	onlyOwner
	swapTokensForEth	Private	✓	
	_transfer	Private	✓	
	addLiquidity	Private	✓	
	_tokenTransfer	Private	✓	CheckDisableFees
	_transferStandard	Private	✓	
	_transferToExcluded	Private	✓	
	_transferFromExcluded	Private	✓	
	_transferBothExcluded	Private	✓	

Inheritance Graph



Flow Graph



Summary

Elons Roadmap 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 stop transactions, manipulate the fees and transfer funds to the team's wallet. The contract can be converted into a honeypot and prevent users from selling if the owner abuses the admin functions. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

Disclaimer

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